EP 35264 (3a)

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30, 17/60	A1	(11) International Publication Number: **WO 98/52131** (43) International Publication Date: 19 November 1998 (19.11.98)

(21) International Application Number: PCT/US98/10116

(22) International Filing Date: 13 May 1998 (13.05.98)

(30) Priority Data:

08/856,375	14 May 1997 (14.05.97)	US
08/856,313	14 May 1997 (14.05.97)	US
08/856,372	14 May 1997 (14.05.97)	US

(71) Applicant: PORTAL INFORMATION NETWORK [-/US]; Suite 200, 20863 Stevens Creek Boulevard, Cupertino, CA 95014 (US).

(72) Inventors: OWENS, Gary, L.; 565 Hope Street, Mountain View, CA 94041 (US). LABUDA, David, S.; 213 Correas Street, Half Moon Bay, CA 94019 (US).

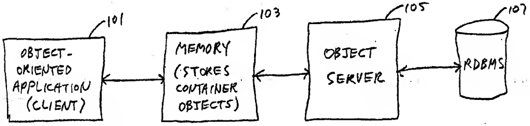
(74) Agent: VAN PELT, Lee; Beyer & Weaver, LLP, P.O. Box 61059, Palo Alto, CA 94306 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR OBJECT ORIENTED STORAGE AND RETRIEVAL OF DATA FROM A RELATIONAL DATABASE TO IMPLEMENT A REAL TIME BILLING SYSTEM



(57) Abstract

Systems and methods for accessing a relational database through an object–oriented querying interface are provided. A class of objects that are to be stored in the relational database are defined. One or more relational database tables are created and a mapping is produced that maps each data member of an object to one or more columns in a relational database table. Additionally, object–oriented paradigms like inheritance may be supported and the allocation of storage for array elements may be deferred until necessary. A container object that allows a user to define new payment resources without requiring the user to redesign a relational database system used for persistent storage of transaction informatiion is also disclosed. A real time billing system for accounts that locks out transaction events when the billing process is underway is also disclosed. When a transaction event is received that should be posted to an account during the billing process, the account is locked.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

METHOD AND APPARATUS FOR OBJECT ORIENTED STORAGE AND RETRIEVAL OF DATA FROM A RELATIONAL DATABASE TO IMPLEMENT A REAL TIME BILLING SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to methods and apparatuses for transferring data to and from a first memory that is organized according to an object-oriented scheme to a second memory that is organized according to a relational database management scheme.

In one embodiment, the invention relates to transferring data to and from a transient storage that is organized according to an object-oriented scheme to a persistent storage that is organized according to a relational database management scheme. In certain embodiments, the relational database in persistent storage is designed by an object server. This includes defining the tables of the relational database as well as the various columns. The object server then stores and retrieves data from the various tables defined in persistent storage according to a hierarchical tree that maps data encapsulated within objects to table locations in the relational database found in persistent storage.

In another embodiment, the invention relates to debiting and crediting existing payment resources or creating new payment resources using an object oriented scheme. Data relating to payment resources is transferred to and from an object server using a container object. The object server interacts with both a transient storage that is organized according to an object-oriented scheme and a persistent storage that is organized according to a relational database management scheme. The relational database in persistent storage is designed by the object server. This includes defining the tables of the relational database as well as the various columns. The object server then stores and retrieves data from the various tables defined in persistent storage according to a hierarchical tree that maps data encapsulated within objects to table locations in the relational database found in persistent storage.

When an event occurs that requires payment from a payment source, or when authorization is sought for an event that will require payment from a payment source, a rating engine analyzes a number of rates that are available, resources that are available, and available credit limits to create a rate stack that determines authorization and charging for a transaction.

In another embodiment, the invention relates to preventing transactions that occur after the end of a billing period but before billing has been completed from being included in bills generated by a real time billing system.

2. Description of the Related Art

There are well known tradeoffs associated with relational database and object -oriented database designs. Relational databases are commonly optimized for fast, efficient searching. This is largely the result of the fact that relational databases are built from a set of tables that contain related columns. The tables are indexed in an efficient manner by the relational database so that searches may be performed in an optimal manner. While organizing information into a complex related set of tables helps speed searching, a thorough knowledge of the tables is required to specify data that is to be retrieved or to specify where data is to be stored. Furthermore, changing the structure of the tables to add a column may require extensive programming and rewriting of existing code. Another problem in many relational database management systems (RDBMSs) is that columns in tables that contain no information or are not used nevertheless take up space in memory.

A standard relational query language, Structured Query Language (SQL) is used to query most popular relational databases. SQL requires that the person who specifies a query know what tables and columns contain the information that is to be compared against the query. For example, in order to look for all customers in a city, the user must know both the name of the table that contains city information and also the name of the column in that table that contains the city information. It is also necessary that the user know the tables that should be joined to accomplish the search. Likewise, in order to store information in the proper column of the proper table, the user must know the name of the table and column in which the information should be stored.

In contrast, it is easier to query, modify and write information to object-oriented databases. Instead of specifying a table and column for storing or retrieving information, related data is encapsulated in an object. The object may be read into memory and all encapsulated data may be readily accessed. Searching, however, is not as efficient as relational database searching. Entire objects are read into memory in order to check the relevant encapsulated data members. Similarly, store operations are performed on entire objects. Thus, this methodology is not very well suited for on-line transaction processing (OLTP) where transaction rates are high but often only portions of the objects are desired.

Attempts to make object-oriented relational databases have for the most part merely added an object-oriented interpretation to a relational database structure. For example, rows in an existing relational database structure may be interpreted as an object, with each column representing an encapsulated data member. This arrangement, however, does not realize the full power of an object-oriented database. For example, inheritance is not supported so subclasses of objects may not be defined. Additionally, the problem of adding data members to objects is

not addressed. Still further, adding a column with no data still allocates large chunk of storage for that column, even if the column is never used.

In view of the foregoing, there is a need for methods and apparatuses for taking advantage of the programming, storage and querying ease of an object-oriented database while enjoying the searching speed of a relational database.

Billing has become increasingly complex in the field of on line transaction processing. In particular, convergent billing, where multiple services are billed to a customer on a single bill has become an important goal of on line transaction processing systems. In such systems, multiple sources of payment may be used to pay for services. For example, a phone company may provide both long distance phone service and internet access and may also handle billing for a certain on line services accessed by a customer. As the customer accesses one service, it may be desired to give the customer free time on one of the other services as an incentive to use both services. Additionally, it may be desired to give the customer some other sort of incentive such as frequent flier miles whenever the customer accesses a certain service. Payment for a given service may then be made from any one of these multiple resources, which must be tracked.

The problem becomes more complex when an authorization event or a payment event occurs. Different payment rates from different sources must be analyzed to determine the rate to be applied. In the case of an authorization, different credit limits applying to different sources must be checked.

One of the most important requirements of such a transaction processing system is the need to add payment resources and rate schemes without extensive reprogramming. Traditional payment systems have not been successful in meeting this requirement. The need for fast, optimized searching of a very large database has militated in favor of relational data base designs for the storage of customer, payment source, and rate information. Relational databases are built from a set of tables that contain related columns. The tables are indexed in an efficient manner by the relational database so that searches may be performed in an optimal manner. While organizing information into a complex related set of tables helps speed searching, a thorough knowledge of the tables is required to specify data that is to be retrieved or to specify where data is to be stored. Creating new data and creating new types of data such as new payment resources generally requires modifying the design of the tables to accommodate the new data and often requires extensive rewriting of existing the system program code. This means that a high level of expertise and familiarity with the database design is required to add payment resources.

Another problem in many relational database management systems (RDBMS's) is that columns in tables that contain no information or are not used nevertheless take up space in memory. Thus, when new payment resources are created, a large amount of memory may be

allocated for tracking the payment source for every customer. Old payment resources that are no longer used also may continue to take up storage space until the database is restructured to remove them.

A standard relational query language, Structured Query Language (SQL) is used to query most popular relational databases. SQL requires that the person who specifies a query know what tables and columns contain the information that is to be compared against the query. For example, in order to look for all customers in a city, the user must know both the name of the table that contains city information and also the name of the column in that table that contains the city information. It is also necessary that the user know the tables that should be joined to accomplish the search and how to join those tables. Likewise, in order to store information in the proper column of the proper table, the user must know the name of the table and column in which the information should be stored.

In contrast, it is easier to query, modify and write information to object-oriented databases. Instead of specifying a table and column for storing or retrieving information, related data is encapsulated in an object. The object may be read into memory and all encapsulated data may be readily accessed. Searching, however, is not as efficient as relational database searching. It would be desirable if an object oriented database could be used to add payment resources and rates and to retrieve information into a transient memory, and if a relational database could be used for persistent storage of data.

In view of the foregoing, there is a need for methods and apparatuses for defining payment resources using an object-oriented database associated with a transient memory and creating a relational data base in persistent memory that stores payment source information. It would also be useful if an efficient system for charging payments at different rates could be developed.

Real time event rating and billing systems offer important advantages over traditional billing systems. In a traditional billing system, transaction events are time stamped and entered into a database. In order to determine an account balance at a certain point in time it is necessary to search the database and retrieve all of the transaction events that occurred within the time period of interest and compute the account balance based on the transaction events found. A disadvantage of such a system is that real time balances are not kept, that is, a search and calculations are required to determine current balances at any point in time. No running balance total is stored in memory.

Real time event rating and billing systems, on the other hand, keep a running total of balances. It may be useful to have such a running total readily available in memory for several reasons. For example, it may be desirable to authorize transactions in real time based on account

balance or apply a rate to a transaction in real time as a function of certain existing balances. Generally, a real time rating system is useful whenever real time account balances are needed.

In most billing systems, bills are generated periodically. Typically, bills are generated for a specific period defined by an opening time and a closing time. It is generally desirable that the close of the billing period be precisely determined and that transaction events outside of the period be excluded from the billing. Generating a clean accounting close is relatively simple for a non-real time system. The definition of the search criteria provides a time cut off that retrieves only transaction events that occur before the close of a billing period. Generating a clean accounting close for a real time event rating and billing system which is continuously adjusting account balances as transaction events occur, on the other hand, presents a more difficult problem. The billing process is not instantaneous and generally takes a finite amount of time to run that in many applications is at least a couple of hours. During that time, if a transaction event occurs, then it may change an account balance before that balance is billed. The transaction would then be improperly recorded since it occurred after the close of the accounting period.

In view of the foregoing, there is a need for methods and apparatuses for providing a clean accounting close for a real time transaction processing system that keeps running balance totals for accounts and does not generate balances using time delimited search criteria.

SUMMARY OF THE INVENTION

Accordingly, the present invention provides an object server that maps data that is represented in transient memory according to an object-oriented scheme to data that is represented in persistent memory according to a relational database scheme. In certain embodiments, the object server generates appropriate tables and columns for a relational database scheme automatically so that an object-oriented scheme generated by a user may be efficiently stored and searched in persistent memory. Preferably, array elements are represented as rows in a table, not as columns so that storage space is not wasted with place holder data.

The present invention also provides a container object that allows a user to define new payment resources without requiring the user to redesign a relational database system used for persistent storage of transaction information. An object server maps data that is represented in transient memory according to an object-oriented scheme to data that is represented in persistent memory according to a relational database scheme. The object server generates appropriate tables and columns for a relational database scheme automatically so that the object-oriented scheme generated by a user may be efficiently stored and searched in persistent memory. Preferably, array elements are represented as rows in a table, not as columns so that storage space is not wasted with place holder data. In certain embodiments, a rating engine is provided that searches available rates and creates a rate stack for the purpose of authorizing transactions and adjusting payment source balances when authorization events or payment events occur.

The present invention also provides an interlock system that locks an account while the account is being billed and then releases the account. When a transaction event occurs during billing, that account is locked and billing for the account is generated immediately. The transaction event is then processed and the account balance is adjusted before the account is unlocked. Subsequent billing for that account during the same period is avoided by checking whether a "last billed" field for the account contains a time that is prior to the closing time for the billing period.

In one embodiment, the invention provides a computer implemented method of storing objects in a relational database comprising the steps of: defining a class of objects that are to be stored in the relational database, the objects of the class having at least one data member; creating at least one relational database table to store the objects of the class; and mapping each data member to at least one column in the at least one relational database table. Subclasses of objects may be defined that inherit the data members of a parent class. The data members for objects of the subclass are typically stored in additional relational database tables.

In another embodiment, the invention provides a computer implemented method of querying a relational database comprising the steps of: receiving a first query that is object-oriented and specifies information about objects of interest; instantiating a query container object that comprises a query template based on the first query, an array for any arguments in the template query, and an array for any results in the template query; utilizing the query container object, translating the first query into a second query in a relational database query language for accessing the specified information about the objects of interest that are stored by a relational database management system; sending the second query to the relational database management system; and receiving the specified information about the objects of interest from the relational database management system. Additionally, a results container object may be instantiated to store the specified information about the objects of interest.

In another embodiment, the invention provides a computer implemented method of deferring allocation of storage for array elements of objects comprising the steps of: receiving a request to instantiate an object of a class where the class has a definition that specifies a default value for each data member of an array element; allocating storage space for the object without storage space for an array element if the instantiation request does not specify an initial value for any of the data members of the array element; receiving a request to modify a data member of the array element; determining if storage space for the array element has been allocated; if storage space for the array element has not been allocated, allocating storage space for the array element and initializing each data member of the array element to the specified default value; and modifying the data member of the array element as specified in the modification request.

In another embodiment, the invention provides a computer system for storing objects in a relational database comprising: an object-oriented application for receiving a definition of a class of objects that are to be stored in the relational database, the objects being stored in a transient storage; a memory for the transient storage of the objects; an object server for retrieving the objects from the memory and issuing statements in a relational database query language to store data of the objects; and a relational database management system for receiving the statements and storing the data of the objects in persistent storage as relational database tables.

In another embodiment, a method of assigning payment resources to an account balance stored in an RDBMS table includes instantiating a balance adjusting container object that includes an account element identification number, a balance element identification number of a balance element that will be adjusted and an adjusting amount and formulating an RDBMS query to determine whether an entry in an account balance table exists that corresponds to the account element identification number and the balance element identification number. If an entry in the account balance table is found, the found entry in the account balance table is adjusted by the adjusting amount. If no entry in the account balance table is found, then an entry is created in the

account balance table with a default balance and the created entry is incremented by the adjusting amount.

In another embodiment, a method of defining a payment resource for inclusion in a real time transaction processing system includes instantiating a balance element creating container object and including data fields in the balance element creating container object including a balance element identification number field and providing a balance element identification number in the balance element identification number field. A new row is added to a payment resources RDBMS table using the information from the balance element creating container object.

In another embodiment, a method of determining a rate for adjusting a real time balance includes receiving a billing event, the billing event including a rate name, a quantity used, an attribute that describes the event, and an account to be billed. A product table is searched to determine all of the products in the account to be billed and a rate table is searched to determine all of the rates that correspond to the rate name and the products in the account to be billed. Rates are eliminated that do not match the attribute that describes the event and the remaining rates are ordered by descending priority to obtain a rate stack. For each rate in descending order of priority, the maximum portion of the quantity used that can be applied using the rate without violating a credit limit is allocated.

In one embodiment, the invention provides a computer implemented method of performing real time billing of accounts comprising the steps of: during close billing of a billing cycle, receiving a transaction event for an account; locking the account; determining if the account has already been billed in the billing cycle; if the account has not already been billed in the billing cycle, close billing the account without billing the transaction event; posting the transaction event to the account; and unlocking the account. The account may be determined that it has not already been billed by checking if the next bill time of the account is in the past. Typically, the transaction event is a billing event.

In another embodiment, the invention provides a computer implemented real time billing system comprising: a processor; a memory coupled to the processor that stores accounts; a billing process operating on the processor that bills accounts at the end of a billing cycle and upon receiving a transaction event for an account, the billing process locks the account, close bills the account without billing the transaction event if the account has not already been billed in the billing cycle, posts the transaction event to the account, and unlocks the account.

These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures which illustrate by way of example the principles of the invention.

PAGE INTENTIONALLY LEFT BLANK

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Figure 1 illustrates an example of a computer system that may be utilized to execute the software of an embodiment of the present invention.

Figure 2 shows a system block diagram of the computer system of Figure 1.

Figure 3 shows a block diagram of an embodiment of the invention which provides an object-oriented interface to objects that are stored by a relational database management system.

Figure 4 shows a graphical hierarchical tree that may be utilized to define a class of objects.

Figure 5A shows a structure of a data block utilized to store container objects and Figure 5B shows a container object for storing an object in transient memory.

Figure 6 illustrates relational database tables that may be generated by an object server of the invention to store objects of the class defined in Figure 4.

Figure 7 shows a process of defining and storing an object in a relational database.

Figure 8 shows a graphical hierarchical tree that may be utilized to define a subclass of objects (e.g., subclass of the class shown in Figure 4).

Figure 9 illustrates the additional relational database tables that may be generated to store the additional data members of objects of the subclass defined in Figure 8.

Figure 10 shows a hierarchical tree that may be maintained to store the relationship of classes and their characteristics.

Figure 11 shows a process of generating a subclass.

Figure 12 shows a process of deleting an object.

Figure 13 shows a process of querying the relational database for information about objects of interest through an object-oriented interface.

Figure 14 illustrates a query container object that may be utilized in querying the relational database through an object-oriented interface.

Figure 15 shows a query container object that may be generated by the invention.

Figs. 16A and 16B show a process of generating SQL searches utilizing a query container object.

Figure 17 shows a process of allocating storage space for an array element of an object only if necessary in order to save storage space.

Figure 18A is an illustration of a container object which stores an account object.

Figure 18B illustrates relational database tables that may be generated by the object server to store objects.

Figure 19 is a process flow diagram illustrating a process for adding a new payment source or balance element to an account.

Figure 203 is a process flow diagram illustrating a process for adding a new balance element type to the on line transaction processing system.

Figure 21 is a block diagram illustrating relational database tables used to implement the rating method used in one embodiment of the present invention.

Figure 22 is a process flow diagram illustrating the process that runs when a billing event occurs for an account.

Figure 23 illustrates some relational database tables that may be stored for the purpose of storing real time account balance information.

Figure 24 shows a regular billing process implemented on a computer that sequentially accesses accounts to generate bills for the accounts.

Figure 25 shows a process that may occur when a billing event occurs while the accounts are being billed.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention. An example of a preferred embodiment is illustrated in the accompanying drawings. While the invention will be described in conjunction with that preferred embodiment, it will be understood that it is not intended to limit the invention to one preferred embodiment. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims along with their full scope of equivalents. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

Figure 1 illustrates an example of a computer system that may be used to execute the software of an embodiment of the present invention. Figure 1 shows a computer system 1 which includes a display 3, screen 5, cabinet 7, keyboard 9, and mouse 11. Mouse 11 may have one or more buttons for interacting with a graphical user interface. Cabinet 7 houses a CD-ROM drive 13, system memory and a hard drive (see Figure 2) which may be utilized to store and retrieve software programs incorporating computer code that implements the present invention, data for use with the present invention, and the like. Although the CD-ROM 15 is shown as an exemplary computer readable storage medium, other computer readable storage media including floppy disks, tape, flash memory, system memory, and hard drives may be utilized.

Figure 2 shows a system block diagram of computer system 1 used to execute the software of an embodiment of the present invention. As in Figure 1, computer system 1 includes monitor 3 and keyboard 9, and mouse 11. Computer system 1 further includes subsystems such as a central processor 51, system memory 53, fixed disk 55 (e.g., hard drive), removable disk 57 (e.g., CD-ROM drive), display adapter 59, sound card 61, speakers 63, and network interface 65. Other computer systems suitable for use with the present invention may include additional or fewer subsystems. For example, another computer system could include more than one processor 51 (i.e., a multi-processor system), or a cache memory.

Arrows such as 67 represent the system bus architecture of computer system 1. However, these arrows are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be utilized to connect the central processor to the system memory and the display adapter. Computer system 1 shown in Figure 2 is but an example of a computer system suitable for use with the present invention. Other configurations

of subsystems suitable for use with the present invention will be readily apparent to one of ordinary skill in the art. t to one of ordinary skill in the art.

Figure 3 shows a block diagram of an object-oriented application that stores objects in a relational database. An object-oriented application 101 allows object-oriented creation, manipulation, and searching of objects. In traditional client-server nomenclature, application 101 is the client and it typically operates on a computer system. The computer system may be similar to the one shown in Figure 1.

As objects are created or accessed by application 101, the objects are stored in memory 103. The memory may be generally thought of as transient storage - meaning that the storage is only temporary and is not the permanent storage of the objects. Typically, memory 103 is the dynamic random access memory of the computer system on which the client computer system operates. Of course, memory 103 is not limited to any specific memory-type as it may be cache memory, flash memory, hard drive, floppy disk, and the like.

An object server 105 provides the interface between the object-oriented scheme and the relational database scheme. The object server is a process that translates object-oriented requests into relational database requests (e.g., SQL). Typically the object server operates on the same computer system as the object-oriented application. However, there is no requirement that the object server operate on the same computer system or at the same location (e.g., the two computer systems may be in communication over a network).

The object server sends relational database requests to a relational database management system (RDBMS) 107. The RDBMS stores data in relational tables with columns in the tables representing data of the same type. Although the RDBMS typically operates on a different computer system than the object server, the RDBMS may operate on the same computer system. In traditional client-server nomenclature, RDBMS 107 is the server. In a preferred embodiment, the RDBMS is from Oracle Corporation, Redwood Shores, California.

Object-oriented environments are intended to shield the implementation from the user. More specifically, in database applications an object-oriented environment hides the details of how the objects are stored. This is in stark contrast to traditional relational database applications where it is generally required for a user to know how data is stored in order to formulate queries on the data. It may be beneficial at this point to illustrate an example of how the present invention may store an object in a relational database scheme.

In object-oriented environments, a class defines a group of objects that share the same characteristics. More specifically in this context, each object (or instance) of a class may have the same data members. An object is created by being instantiated as a member of a class.

Figure 4 shows a graphical hierarchical tree that may be utilized to define a class of objects. A graphical hierarchical tree 151 defines a class named "Account" which three data members. In this example, a class for an accounting system will be defined. However, this example is intended to aid the readers understanding of the invention and does not limit the invention to any specific embodiment.

Data member "Name" represents the name of a customer. Data member "Last Billed" represents the date that the customer was last sent a bill. Lastly, data member "Balance" is an array where each element of the array may store three data members. The graphical hierarchical tree indicates that "Balance" is an array by the brackets indicated by arrow 153. The minus sign in box 155 may be activated to collapse the data members of Balance as is commonly done in conventional graphical user interfaces (GUIs).

Each element of the array Balance may include a data member "Name" which represents the currency of the units of the balance, a data member "Current Balance" which represents the current balance, and a data member "Credit Limit" which represents the credit limit. Although this is a very simple example, it illustrates many of the advantages of object-oriented data storage. For example, it is easy for a user to define a class of objects. The user may use pull-down menus to select a type of data member (e.g., integer) . Then the user may drag and drop the new data member on the graphical hierarchical tree at the desired location.

A significant advantage of an object-oriented is inheritance. Inheritance allows one class, the subclass, to inherit or receive all the characteristics of a higher or parent class. Inheritance allows a user to tailor new classes off of an existing class, therefore resulting in a reuse of resources. An example of a subclass of Account will be described in more detail in reference to Figs. 8-10.

A hierarchical tree provides a hierarchy for the data members. Accordingly, the system does not have trouble distinguishing Name which is a data member of each Account object and Name which is a data member of each Balance array element.

In order to provide a thorough understanding of the invention, Figs. 5A and 5B show a container object for storing an object in transient memory. However, it should be understood that the actual implementation of the container object is not apparent to the user. Figure 5A shows the structure of a data block that may be utilized to store a container object.

A data block 201 includes four fields for storing the following information:

Name - indicates the name of the data in the data block

Type - indicates the data type of the data in the data block

Value - stores the value of the data in the data block

Element Id - indicates the id of an element of an array

In order to better understand the data blocks, one should refer to Figure 5B which shows a container object 203 which stores an object of the class defined in Figure 4. The container object includes a main container 205 and two subcontainers 207 and 209.

Main container 205 includes a header 211 which is for storing any information about the main container (e.g., number of subsequent data blocks and their location in memory). Following the header, there are one or more data blocks that have the same structure as the data block shown in Figure 5A.

Referring to a data block 213, the Name of the data block is <Sys Id> indicating that this data block stores a system id for the object, which may be thought of as the name of the object from the system's point of view. In a preferred embodiment, the Sys Id includes a type string, id number, database number, and revision number.

In data block 213, the Type is "Sys Id" indicating that this data block identifies an object. The Value of the data block stores 3456 which is a number which will be utilized to join relational tables storing data members for this object. Accordingly, this number will be referred to as the "Id" when discussing the relational database tables storing the object. As mentioned in the preceding paragraph, the Sys Id may be a compound data type including an id number. Thus, in some embodiments, the Id (e.g., 3456) is a portion of the Sys Id. Lastly, the Element Id is "N/A" (not applicable) as this data block does not store an element of an array. Thus, it should not matter what is stored in the Element Id field.

Although data block 213 contains the same fields as the other blocks, it is a unique block as it identifies an object. Accordingly, a better understanding of the data blocks will be achieved from a detailed discussion of the subsequent data blocks.

A data block 215 stores a data member of the object. The Name field contains "Name" which is the name of the first data member of the class that is defined in Figure 4. The Type of the data block is "String" which indicates the Value field stores a string which is shown as "John Doe." As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 217 stores another data member of the object. The Name field contains "Last Billed" which is the name of the second data member of the class (see Figure 4). The Type

of the data block is "Date" which indicates the Value field stores a date which is shown as a string for April 1, 1997. As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 219 stores an element of the array "Balance" utilizing subcontainer 207. The Name field contains "Balance" which is the third data member (an array) of the class (see Figure 4). The Type of the data block is "Array" which indicates the Value field stores an element of an array. The Value field stores an element of the array by storing a pointer to subcontainer 207 which stores the data members of the element. The Element Id field contains the number 840, which according to the International Standards Organization (ISO) specifies U.S. dollars. Although in this particular example the Element Id field has a specific meaning in addition to identifying an array element, typically the Element Id field acts to identify the array element. The Element Id field will be utilized to join relational tables storing data members for this object.

Subcontainer 207 contains three data blocks which indicate that this object has a Current Balance of 500 U.S. Dollars and a Credit Limit of 10,000 U.S. Dollars. The first data block indicates the currency of the other two data blocks.

A data block 221 stores another element of the array "Balance" utilizing subcontainer 209. Thus, each element of the array may include a data block and a subcontainer. Substructures are implemented similar to arrays and may be thought of as an array with one element (the Element Id may be inapplicable though). Of course, there is no limit to the complexity of the objects stored under this implementation. For example, an array may contain an array, which contains an array, and so on.

Although the data blocks were shown in the same order as the data members were represented in the graphical hierarchical tree defining the class shown in Figure 4, it is not necessary that the data blocks be stored in any particular order. For example, when an element is added to an array, the data block for the new element may be added to the end of the main container.

Putting the container object shown in Figure 5B into perspective, the container object is typically stored in memory 103 shown in Figure 3. Therefore, the container objects are the method of communicating data (along with API calls) between the object-oriented application and the object server. Now it may be beneficial to describe the way the object will be stored in the relational database as relational tables.

Figure 6 illustrates relational database tables that may be generated by the object server to store objects. A relational database table (Account_T) 251 is the main relational table for the

objects. As shown, the table includes columns entitled "Id," "Name" and "Last_Billed." The Id refers to a number that will be utilized to identify this object in the relational database and therefore, join relational database tables. In some embodiments, a Sys_Id column may be utilized in place of the Id column where the value in the Sys_Id column is compound data type to identify the object where Id is included in the Sys_Id. The Name and Last_Billed columns store data members of the object (see Figure 4).

A relational database table (Account_Balance_T) 253 stores elements of the array "Balance." As shown, the table includes a column entitled "Id" which stores the id of the object for which the data in this table belongs. As one familiar with relational databases will recognize, the Id will be utilized to join relational database tables 251 and 253. Table 253 also includes a column entitled "Element_Id" which designates the element id of this element. In the instant case, the element id designates the currency of the data in the balance element. Lastly. table 253 includes columns entitled "Name," "Current_Balance" and "Credit_Limit" which may be data members of an element of the array Balance (see Figure 4). In some embodiments, the Name column is located in a different relational database table but it is shown here in table 253 for simplicity.

Additionally, the value in the Element_Id column is utilized to identifyBalance array elements. Each Balance array element is stored in the relational database as a row in a relational table. Accordingly, storage space need only be allocated for those array elements that have been declared. In some embodiments, the allocation of storage space for array elements is deferred until needed in order to save storage space. This process will be described in more detail in reference to Figure 17.

It should be readily apparent that by storing data members of objects in relational database tables, one may use conventional relational database management systems to query the relational database tables. Other features of the invention that will be described in more detail below are that array elements are easily added to an object by the addition of another row of a relational database table, subclasses may be defined which inherit the data members of a higher or parent class, and array elements of objects need not be allocated until actually utilized (this will also be called "lazy allocation of array elements").

Figure 7 shows a process of defining and storing an object in the relational database. At step 301, the user defines a class for the object. The class may be parent class or a subclass that inherits characteristics from another class. For the moment, assume the user defines the class shown Figure 4.

The object-oriented application analyzes the class definition and generates a hierarchical tree for parsing at step 303. The hierarchical tree includes information for mapping an object's

data members to columns in a relational database table. Accordingly, the hierarchical tree is parsed in order to map from the object-oriented scheme to the relational database scheme. The hierarchical tree also includes information about inheritance between or among classes as will be shown in Figure 10.

At step 305, the object-oriented application instructs the object server to generate the requisite relational database tables. The object server then sends SQL calls to the RDBMS to generate the specified tables. At this point, the relational database tables have been generated but they do not contain any data.

In order to store an object in the relational database, the object-oriented application instantiates a container object at step 307. The container object will store the data members of the object while it is in transient memory. The user sets the value of data members of the object at step 309.

When a class is being defined, the user may specify certain characteristics of the data members. These characteristics are stored in the hierarchical tree that is utilized to map between the object-oriented scheme and the relational database scheme. For example, the user may specify that a data member is mandatory for an object of the class. The system then verifies that every object created of that class contains a value for that data member. Conversely, the user may specify a data member is optional so that the data member is not required for every object of the class.

The user may also specify that data members have default values. These default values will also be stored in the hierarchical tree and may be utilized to initialize an optional data member if no initial value is specified. If no value is specified for a mandatory data member, the user is informed that this is an error.

Additionally, default values may be utilized for the lazy or late allocation of array elements. An array element with a default value for each data member of the array element need not be allocated storage space in the container object (and therefore the relational database tables) until the array element is accessed or modified. Thus, only when an array element is actually needed is the storage space allocated. This feature of the invention provides significant storage savings.

At step 311, the object-oriented application instructs the object server to store the object in the relational database. The application sends an API call to store an object along with a reference or pointer to the container object. The data in the container object includes the values for the data members of the object.

The object server references the container object and generates SQL calls to the RDBMS to store the data in the appropriate relational database tables. The RDBMS will generate the necessary new rows and store the data in the appropriate columns of the relational database tables.

The above has described a simple example where the defined class is a highest level class (i.e., is not a subclass so it does not inherit characteristics from a parent class). One of the powerful features of the invention is that subclasses may be defined and objects of those classes inherit characteristics (e.g., data members) from a parent class and may be stored in the relational database.

Figure 8 shows a graphical hierarchical tree that may be utilized to define a subclass of objects. A graphical hierarchical tree 351 is shown that is divided into two halves by a dashed line 353. The top half of the graphical hierarchical tree is the same as shown in Figure 4. Thus, the subclass being defined inherits the characteristics of the class "Account."

As shown, the subclass adds a new data member "Address" which is a substructure including a data member named "Street," a data member named "City" and a data member named "Phone" which is an array. Each element of the array Phone may have a data member named "Type" and a data member named "Number." As an example, the Type data member may indicate the Number of the Phone element is a home phone number.

The container object for storing an object of the subclass may look similar to the one shown in Figure 5B with an additional data block in the main container for the Address that points to a subcontainer that stores the data members for the Address. This subcontainer will also have a data block that points to a subcontainer for each element of the array Phone.

Figure 9 illustrates the additional relational database tables that may be generated to store the additional data members of objects of the subclass. A relational database table (Account_Address_T) 401 stores data for the two nonarray data members of Address. As shown, a single address has been stored in the relational table. The Id column indicates that this is an address for John Doe as it has the same id number. The relational database system will utilize this column to perform joins on the relational database tables.

A relational database table (Account_Address_Phone_T) 403 stores data for each element of the array Phone. As shown, there are two elements present with element being a row in the relational database table. Each row includes an Id and an Element_Id, in addition to the data member columns. Since each element of an array is represented as a row in a relational table, it is easy to add new elements to the array as new rows since RDBMS are designed to add new data in this fashion.

Although adding new rows to relational database tables is quite easy, this is not the case with adding new columns. For this reason, in some embodiments, the data members defined in subclasses are constrained to be stored as substructures and arrays by the RDBMS. As mentioned earlier, substructures may be thought of conceptually as a one element array. Both substructures and arrays are represented in the relational database as new tables.

Figure 10 shows a simple representation of the hierarchical tree that may be maintained to store the relationship of the classes and their characteristics. A hierarchical tree 451 includes a node for each related class and subclass. The topmost node A represents the parent class. The nodes X and Y are subclasses of the parent class and therefore inherit characteristics of the parent class. Node Z represent a subclass of the class X so it inherits characteristics of both the parent and grandparent classes.

At each node is stored information to map each data member of the class to one or more columns in the relational database tables. For example, a mapping from the data member Current Balance of the subclass to the relational database column Account_Balance_T.Current_Balance (see Figure 6). Additionally, each node will store other characteristics of the data members of objects of the class (e.g., default values).

The hierarchical tree allows a user to define a class hierarchy that is arbitrarily complex and each class itself may be arbitrarily complex. For example, at any node in the hierarchical tree, there may be any number of data members, structures, arrays or nesting of these data types (limited only by the capacity of the computer system or software).

Although Figure 7 shows a process for both defining a class and storing an object of the class, these operations may be performed separately as will be demonstrated by a discussion of Figure 11 which shows a process of generating a subclass. At step 501, the user defines a subclass as was discussed in reference to Figure 8. The object-oriented application generates a hierarchical tree for parsing. This hierarchical tree includes characteristics of the subclass and the parent class from which it inherits.

At step 505, the object-oriented application determines a difference between the hierarchical tree for this subclass and the hierarchical tree for its immediate parent class. Conceptually, this would identify the new data members as is shown on the bottom half of Figure 8.

The object-oriented application issues an API call to the object server generate the new tables for the differences between the hierarchical tree for the parent class and subclass at step 507. The object server issues the appropriate SQL calls to generate the additional relational database tables for objects of the subclass. Data members for the objects which are inherited

from the parent class(es) will be stored in the relational database tables that were generated for each class. Thus, data members that are common to related objects will be stored in the same relational database tables. This provides a number of advantages including the following.

An advantage of the invention is that one is not limited to any one class during queries. For example, one may search for all objects that were billed last on 4/1/1997. The RDBMS and the object server may return objects that, although related, are not the same class. Some objects may be of a parent class while other objects may be of a subclass, or a subclass of a subclass. The object server returns a list of objects in a container object that satisfy the query.

There are numerous API calls that may be issued to the object server. A detailed description of all the API calls is not necessary to understand the invention. However, it may be beneficial to illustrate a process of utilizing one of the available API calls. The API call that will be discussed deletes an object from the relational database.

Figure 12 shows a process of deleting an object. At step 551, the object-oriented application instantiates a container object in transient memory. It is presumed that it is already known which object should be deleted (e.g., results from a search). The object to be deleted will be identified by the Id (or Sys Id), which is the id given to the object by the system.

At step 553, the object-oriented application inserts the Id of the object to be deleted into the container object. The application then issues an API call to the object server at step 555 instructing it to delete the object identified by the container object.

The object server then verifies that the Id in the container object specifies a valid object at step 557. After verification, the object server generates the appropriate SQL calls to delete the data in the relational database at step 559. In a preferred embodiment, the multiple SQL calls that may be necessary to delete an object from the relational database tables are performed as one atomic operation. In other words, the multiple SQL calls are performed as one operation conceptually so the user is not able to access a partially deleted object in the relational database. The one or more SQL calls to delete the oject are received by the RDBMS and executed to perform the deletion .

The invention utilizes the power of relational database management systems to access objects that are stored in the relational tables. Figure 13 shows a process of querying the relational database for information about objects of interest through an object-oriented interface. Before describing the figure, it may be beneficial to review some fundamentals of relational databases.

One of the most popular relational database query languages is SQL. The basic format of an SQL query is the following:

SELECT {columns} FROM {tables} WHERE {conditions}

As an example, suppose using the sample database described in Figs. 4-6 that one wants to know the names of people who have accounts that were last billed on April 1, 1997 and have a current balance greater than $100. An SQL query for this information may resemble the following:

SELECT Acount_T.Name FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100 and Account_T.Id = Account_Balance_T.Id

As should be apparent, the user is required to know the way the data is stored in the relational database to form a correct query. For example, the last condition that specifies that the ids of the two tables are the same specifies how to link the two tables and is called a "join" operation.

Referring again to Figure 13, the object-oriented application receives an object-oriented query at step 601. The object-oriented query will typically specify values of data members of objects of interest. The application instantiates a container object at step 603. As with any flowcharts depicted herein, there is no implied order of the steps simply by the way they are presented.

In general, the application exchanges data with the object server through container objects stored in transient memory. Since this container object is for performing a query, we will call it a query container object. Figure 14 illustrates a query container object that may be utilized in querying the relational database. As shown, a query container object 651 includes a query template, args (or arguments) array and a results array. The query template contains a query with gaps that are filled in by the args array and results array. The query container object will be described in more detail in reference to Figure 15.

At step 607, the object-oriented application instructs the object server to perform the query. The application sends an API call to the object server that references the query container object. The object server analyzes the information in the search object container and generates SQL calls (or queries) for the RDBMS at step 609. This process will be described in more detail in reference to Figs. 16A and 16B. After step 609, the object-oriented query has been translated into a relational database query.

In order to perform the query, the object server may send multiple SQL queries to the RDBMS. Once the object server receives the desired results, the object server instantiates a new container object and stores the results of the query in the container object. Although the results may be placed in the search object container, instantiating a new container object allows the original search object container to be unmodified which may be preferable.

At step 613, the object-oriented application retrieves the results of the query from the container object that stores the results in transient memory and displays the results. The results may be displayed any number of ways but are preferably displayed in an object-oriented fashion.

When a user specifies a query in an object-oriented system, the user specifies the desired data members of the objects of interest (or the whole object) along with the conditions that determine the objects of interest. Thus, the relational database query described above (i.e., SELECT Acount_T.Name FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100 and Account_T.Id = Account_Balance_T.Id) in an object-oriented environment would conceptually be "retrieve the Name data member of all objects of class Account where the Last Billed is equal to April 1, 1997 and the Current Balance is greater than 100 dollars." The form of the object-oriented query may vary and the form of some embodiments will be described in the following paragraphs. However, it is important to notice that with an object-oriented query, the user is not required to know exactly how the objects are stored in memory. This is in stark contrast to conventional relational database systems as is illustrated by the SQL query.

Once the user has entered an object-oriented query, the object-oriented application fills in the query container object. Thus, the application inserts the query template, args array and results array into the query container object. The query template is an SQL-like query that includes object-oriented information as follows:

SELECT { } FROM {class} WHERE {conditions without joins}

The query container object includes a results array which is an array which holds the data members and/or objects that should be returned from the relational database. Since the results array includes what is typically included in the SELECT clause, this clause may just have a place holder like 'X' as in some embodiments.

The FROM clause specifies the class of objects that should be searched. If a parent class is specified, then subclasses will satisfy the query also. Lastly, the WHERE clause specifies conditions like Data_Member1 = Value1. Data_Member1 and Value1 are determined from the first element in the args array. Thus, the first element in the args array would specify an element

of the Name "Last Billed," Type "Date" and a Value of "4/1/1997." The args array fills in the missing values of the query template which may be as follows:

SELECT X FROM Person WHERE Data_Member1 = Value1 and Data_Member2 > Value2

The SELECT clause has an X as a place holder so the query is easier to parse and looks better, but the results array will specify the data members and/or objects requested. As indicated above, the WHERE clause will be filled in from the args array but it should be noted that there are no joins specified.

Figure 15 shows a query container object that may be generated by the invention for the above search. A query container object 701 includes a main container 703 and subcontainers 705, 707, 709, and 711. A data block 713 in the main container stores the query template as discussed above. Data blocks 715 and 717 store the pointers to the elements of the args array. Data block 715 points to subcontainer 705 which includes a data block 719. Data block 719 includes the information to fill in one of the conditions of the query template (the operator '=' is already in the query template). The Name field indicates that Data_Member1 is the "Last Billed" data member and the Value field indicates that Value1 is "4/1/1997" (see Figure 5A for details on the data blocks).

Similarly, data block 717 points to subcontainer 707 which includes a data block 721. Data block 721 points to an array element for the Balance array. Data block 721 points to a data block 723 which includes the information to fill in another of the conditions of the query template (the operator '>' is already in the query template). The Name field indicates that Data_Member2 is the "Current Balance" data member and the Value field indicates that Value2 is 100.

A data block 725 stores an element of the results array. Data block 725 points to subcontainer 711 which includes a data block 727 that indicates the results the query requests. As you may recall, the user wanted the name of the individual. Data block 727 indicates this by having a Name field that specifies the "Name" data member and a Value field that is blank indicating this information should be the results obtained by the query.

Figs. 16A and 16B show a process of generating SQL searches utilizing a query container object (see also step 609 in Figure 13). In some embodiments of the invention, the process shown is performed by the object server. As will be described, what may appear to be a single query may be implemented in multiple SQL queries. However, the invention may ensure that the minimum joins that are necessary are performed as joins are a relatively expensive relational database operation.

The object server will be constructing SQL queries for the RDBMS that retrieve the data of interest. In order to aid the reader's understanding, the process will be described as implementing the query that has been described above where one is searching for the Name data member of objects of class Account that were Last Billed on April 1, 1997 and have a Current Balance greater than 100 dollars.

At step 751, the column for the object Id is put in the SELECT clause. The object server utilizes the hierarchical tree to map the object Id to the associated column in the relational database. The following shows the SQL query that has thus far been generated:

SELECT Account_T.Id FROM WHERE

The object server identifies relational database tables that will be needed to satisfy this query and inserts the tables in the FROM clause at step 753. The object server first identifies all the relational database tables that contain columns to which the data members in the args array are mapped. This involves an analysis of the hierarchical tree and the args array in the query container object. Additionally, the object server determines if any additional tables that are needed for any joins that may be necessary to link the columns in different tables. The following shows the SQL query with the requisite tables:

SELECT Account_T.Id FROM Account_T, Account_Balance_T WHERE

At step 755, the object server inserts conditions from the query specified by the query container object in the WHERE clause. The object server identifies the data members in the args array and uses the hierarchical tree to map the data members to the appropriate columns in the relational database tables. The comparators (e.g., '=' and '>' in this example) are retrieved from the query template. The values to complete the conditions are retrieved from the args array. The following shows the SQL query with the conditions for this example:

SELECT Account_T.Id FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100

Since SQL specifies that joins are explicitly placed in the query, the object server then uses the hierarchical tree to identify the requisite joins and insert the joins in the WHERE clause at step 757. The process of identifying the requisite joins is similar to the process of identifying all the relational tables for the query at step 753. Accordingly, the steps may be performed at the same time.

The object server identifies the joins that are required by parsing the hierarchical tree to determine the relational database tables (e.g., column) to which each of the data members specified in the query template and args array. For each of the different tables, the object server

has to generate a database join to link the two tables (unless that join has already been identified). The following shows the SQL query with the necessary join:

SELECT Account_T.Id FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100 and Account_T.Id = Account_Balance_T.Id

Although in this simple example it is not required, the object server may need to generate multiple joins to link two tables and the joins may include tables that are not immediately apparent from an inspection of the query template and the args array. For example, while the object server is parsing the hierarchical tree, it may identify a table "between" two previously identified tables, therefore creating joins to that intermediate table. By analyzing the hierarchical tree, the invention is able to generate an SQL query that utilizes the minimum number of tables and joins.

Once the SQL query is generated, the object server issues the query to the RDBMS at step 759. The SQL query produced requests the object Ids of all objects of the class Account that satisfy the query. The desired Name data members will identified in a separate SQL query described in Figure 16B.

Referring now to Figure 16B, the object server receives from the RDBMS the object Ids of the objects that satisfy the SQL query at step 761. The objects that satisfy the query may include objects that are of subclasses of the Account class. Thus, the invention fully supports inheritance during searching. The user could also have specified a subclass and the SQL query would have joined in a table that would select only objects of that subclass (and those that inherit from that subclass).

At this point, the object server now knows which objects satisfy the query but the query container object has a results array that specifies the Name data member is desired. Therefore, the object server generates a second SQL query to retrieve the desired data members.

At step 763, the object server inserts the desired columns identified in the results array in the SELECT clause. In this example, the results array specifies the Name data member of the class Account. The object server uses the hierarchical tree to map the data member to the appropriate column or columns in the relational database and produces the following initial SQL query:

SELECT Account_T.Name FROM WHERE

The object server identifies the tables for this query and inserts them in the FROM clause at step 765. With this simple example, the only relational database table that is required is Account_T so the SQL query becomes the following:

SELECT Account_T.Name FROM Account_T WHERE

At step 767, the object server inserts the desired object Ids into the WHERE clause. Although many objects (or none) may satisfy a query, assume that the only one that does has an Id data member of 3456. The SQL query will then become the following:

SELECT Account_T.Name FROM Account_T WHERE Account_T.Id = 3456

Since the first SQL query found objects that satisfied the query, the same conditions do not have to be put in the second SQL query. At step 769, the object server uses the hierarchical tree to identify any joins that may be required to be inserted in the WHERE clause. With this example, no joins are necessary so the SQL query remains unchanged. The object server then issues the SQL query to the RDBMS at step 771.

As mentioned earlier, one of the advantages of the invention is that a user is able to specify characteristics of data members including default values. Default values may be used to defer allocation of storage space (e.g., memory) for array elements ("lazy allocation"). The default values for array elements are stored in the hierarchical tree that is utilized for parsing and the following discusses how allocation of storage space is deferred.

Figure 17 shows a process of allocating storage space for an array element of an object only if necessary in order to save storage space. If the class (or subclass) definition defines a default value for each data member of an array element and an object of that class is instantiated without specifying a value for a data member of an array element, the system does not allocate storage for the array element. Subsequently, when the system receives an instruction to access or modify a data member of the array element at 801, the system checks if the array element has been allocated as shown at step 803.

If the array element has not yet been allocated storage space, the system allocates storage space for the array element and assigns each data member of the array the appropriate default value from the hierarchical tree at step 805. Once the system has verified that storage space for the array element has been allocated, the system may access or modify the data member (e.g., increment) as instructed at step 807.

As was described in Figure 5A, information is transferred by a user to and from the object server using container objects. Each object within a container object includes a number of data blocks. A data block includes four fields for storing the following information:

Name - indicates the name of the data in the data block

Type - indicates the data type of the data in the data block

Value - stores the value of the data in the data block

Element Id - indicates the id of an element of an array

Figure 18A is an illustration of a container object 1803 which stores an account object. The account balances for each payment source are represented as a balance array, as is further described below. The container object includes a main container 1805. Main container 1805 includes a header 1811 which stores information about the main container (e.g., number of subsequent data blocks and memory location). Following the header, there are one or more data blocks that provide account information.

Referring to a data block 1813, the Name of the data block is <Sys Id> indicating that this data block stores a system id for the object, which may be thought of as the name of the object from the system's point of view. In a preferred embodiment, the Sys Id includes a type string, id number, database number, and revision number.

In data block 1813, the Type is "Sys Id" indicating that this data block identifies an object. The Value of the data block stores 3456 which is a number which will be utilized to join relational tables storing data members for this object. Accordingly, this number will be referred to as the "Id" when discussing the relational database tables storing the object. Lastly, the Element Id is "N/A" (not applicable) as this data block does not store an element of an array. Thus, it should not matter what is stored in the Element Id field.

A data block 1815 stores a data member of the object. The Name field contains "Name" which is the name of the first data member of the class. The Type of the data block is "String" which indicates the Value field stores a string which is shown as "John Doe." As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 1817 stores another data member of the object. The Name field contains "Last Billed" which is the name of the second data member of the class. The Type of the data block is "Date" which indicates the Value field stores a date which is shown as a string for April 1, 1997. As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 1819 stores an element of the array "Balance" utilizing subcontainer 1807. The Name field contains "Balance" which is an array because the type field of the data block indicates that it is an array. The balance array is used to keep track of each different type of payment source for the account. Instead of containing an actual value or set of values as would normally be the case for an array, the value field contains a memory pointer to a subcontainer 1807 which stores the data for the element of the array balance that is represented by data block

1819. The Element Id field contains the number 840. In this example, 840 functions in two ways. First, 840 is the array index. Thus the array element balance (840) is distinguished from and array element balance (920). 840 also happens, according to the International Standards Organization (ISO), to specify U.S. dollars. The Element Id field is used to join relational tables storing data members for this object.

This special relationship of arrays to memory is significantly different from the way arrays are generally stored in memory. Generally, all of the elements of an array are assigned locations in memory and the index of the array points to the specific memory location that is assigned to the element of the array that corresponds to the index. If certain elements of the array contain no data, then those memory locations are reserved, but blank. This can result in a very large amount of memory being allocated or reserved in order to add a single element to an array.

This can be clearly seen in the present example, where every account has a balance array. In a typical on line transaction processing system, there may be millions of accounts. Additionally, each balance array as shown is actually an array of structures. For every balance element, there exists a current balance, and a credit limit. Without the structure shown in Figure 18A, adding a new payment source, and therefore a new balance element, would be extremely expensive from a memory standpoint. Each balance structure (of which there is one for every account) would have to have the additional fields added to it, at great cost in memory.

In contrast, the scheme shown in Figure 18A provides a flexible or "lazy" allocation of memory to elements in the array. For each account, only those array elements that represent payment resources included within that account are ever assigned memory. Each balance array element is included in the account container as a data block and the value of the data block points to a memory location of a container that includes the name, current balance, and credit limit for the balance array element. The amount of allocated memory that is saved is large, especially if a large number of accounts exist and certain balance elements are shared by a relatively small number of account holders, and/or if a large number of payment resources exist..

Subcontainer 1807 contains two data blocks which indicate that balance array element 840 for the account in container 1805 has a Current Balance of 500 U.S. Dollars and a Credit Limit of 10,000 U.S. Dollars. The name of the currency of the two data blocks is indicated by the array element index, which is also called the balance element ID. Other nonstandard indices may be used to track resources such as frequent flier miles. A data block 1821 stores another element of the array "Balance" utilizing subcontainer 1809.

Putting the container object shown in Figure 18A into perspective, the container object is typically stored in memory 103 shown in Figure 20. Therefore, the container objects are the method of communicating data (along with API calls) between the object-oriented application and

the object server. As is described in detail in U.S Patent Application Attorney Docket No. PORTP001 filed May 14, 1997, an hierarchical list is used to map the data included in data objects placed in containers to the various fields of a relational database. Now it may be beneficial to describe the way that payment resources are stored in the relational database as related tables.

Figure 18B illustrates relational database tables that may be generated by the object server to store objects. A relational database table (Account_T) 1851 is the main relational table for the objects. As shown, the table includes columns entitled "Sys_Id," "Id," "Name" and "Last_Billed." The Sys_Id refers to an identification of the object given by the system. The Id refers to a number that will be utilized to identify this object in the relational database and therefore, join relational database tables. The Name and Last_Billed columns store data members of the object.

A relational database table (Account_Balance_T) 1853 stores elements of the array "Balance." As shown, the table includes a column entitled "Id" which stores the id of the account object for which the data in this table belongs. As one familiar with relational databases will recognize, the Id will be utilized to join relational database tables 1851 and 1853. The Id may also be utilized to join relational database table 1853 with other relational database tables that are included in the database, such as an account system. Table 1853 also includes a column entitled "Element_Id" which designates the element id of this element. In the instant case, the element Id indicates which element of the balance element array is stored in the row and also designates the currency of the data in the balance element. Lastly, table 1853 includes columns entitled "Current_Balance" and "Credit_Limit" which are the data members of each element of the array Balance in this example.

It should be readily apparent that by storing data members of objects in relational database tables, it is possible to use conventional relational database management systems to query the relational database tables. Thus, the benefits of relational database system searching are enjoyed while the object oriented interface simplifies adding payment resources and adjusting payment source balances.

A relational database table (Resources_T) 1854 keeps track of the various payment resources that have been defined for the system. As shown, the table includes a column for each balance element ID. The resource name for each balance element is also included in a separate column, as is a graphical user interface (GUI) name. The GUI name is used in systems where special GUI information is provided to the user in certain circumstances. Another column defines the rounding characteristics for each balance element.

Adding a new type of payment source to the system involves adding a new array element to the balance element array. This is accomplished in memory by adding a row to table 1854. Adding an existing payment source to an account object involves adding a new row to table 1853. The processes for adding a new type of payment source and for adding an existing payment source to an account object are described in further detain in Figures 19 and 23. An important feature of the memory system described above is that memory is not allocated to arrays such as the balance element array until it is needed. This is the so-called "lazy allocation" of memory.

Figure 19 is a process flow diagram illustrating a process for adding a new payment source or balance element to an account. The process starts at 1900. In a step 1902, a balance adjusting container object is instantiated that contains an account ID number, and one or more balance element ID's of the balance element that is to be adjusted, and an adjustment amount for one or more balances. (Note that a single balance element, such as dollars may have several balances associated with it such as a credit balance and a credit limit.) Next, in a step 1904, the container object is sent to the object server. In a step 1906, the object server formulates a query to find the one or more entries in Account_Balance_T that corresponds to the account ID number and balance element specified in the balance adjusting container object. If an entry is not found, then the object server formulates a query for the table Resources_T to see if the balance element exists in Resources_T. If the balance element does not exist in Resources_T, then it is undefined system wide and so an error message is sent in a step 1907 and then the process ends at 1920. If the balance element exists in Resources_T, then control is transferred to a step 1908 and a row is added in Account_Balances_T with default values in the columns and control is transferred to a step 1910 where the newly created entry balances are incremented by the amounts specified in the balance adjusting container object. If an entry is found in Account_Balance_T in step 1906, then control is transferred directly to step 1910 and the found entry balances are incremented by the amount specified in the balance adjusting container object, which may be either a positive or a negative number Once step 1910 is completed, the process ends at 1920.

The process described in Figure 19 is considerably simplified from the standpoint of the application in that it works whether or not a particular balance exists for an account. If the balance does not exist, it is simply added with a default value of zero and then is incremented in step 1910.

Figure 20 is a process flow diagram illustrating a process for adding a new balance element type to the on line transaction processing system. The process starts at 2000. In a step 2002, a balance element type creating container is instantiated. In a step 2004, A balance element ID, a resource name, a GUI name, and a rounding amount are added to the balance element type creating container as data objects. In a step 2005, the object server checks to make sure that all

of the fields required to add a new balance element type are in the container. If not, then an error message is generated. In a step 2006, the object server then adds a new row to the Resources_T table using the information from the balance element type creating container object. The process ends at 2008.

Thus, adding a new type of payment source to the system is done by instantiating a balance element type creating container object and providing the necessary information in data blocks contained in the balance element type creating container object. Adding a new type of payment source to an account is accomplished by instantiating a balance adjusting container object. and providing the necessary information in the data blocks contained in the balance adjusting container object.

So far, a system and method for defining payment resources and assigning payment resources (which will now be referred to as "resources") to accounts has been described. Now, a system and method for increasing and decreasing the balances of different payment resources defined for an account according to a flexible rating method will be discussed.

The following definitions apply to the rating method described below:

Product - A product is an asset that is bought by a customer that may influence or determine amount of resources that the customer will be charged for consuming certain rated quantities. An example of a product would be a certain calling plan that entitles the user to pay 1.00 per hour for a certain rated quantity. Another product could be a plan that entitles the user to pay 2.00 per hour for the same rated quantity.

Rate - A rate is something that determines the cost of a quantity, that is a rate determines how a quantity will impact customer balances when it is consumed. For example, a rate used by an internet service provider may be "type A connect time". The rate determines the cost of a quantity of hours or other consumable according to its impact on various balances.

Figure 21 is a block diagram illustrating relational database tables used to implement the rating method used in one embodiment of the present invention. A Product_t table 2102 contains a list of each product that is available. Product_t table 2102 includes Product ID column that identifies the product, a Name column that provides the name of the product, and a number of attribute columns that may describe things about the product such as when the product may be purchased or the type of customer who is eligible to purchase the product.

A Rate_t table 2104 contains a list of all of the rates that may be charged for a quantity. A Rate_t table 2104 includes a Rate ID column that identifies the rate, a product ID column that identifies the product associated with the rate, and a rate name column that gives the name of the

rate. It should be noted that rate names are generally not unique. the same rate name may be repeated for many rate ID's. Rate_t table 2104 also includes a priority column that indicates the priority of each rate, a default flag column that indicates whether each rate is eligible as a default rate, and an attribute column that indicates such attributes of the rate such the time of day when it is applicable.

An ACCT_t table 2106 contains a list of the accounts. ACCT_t table 2106 includes an Account ID column that identifies the account and various account attributes such as the customer name and when the customer was last billed.

An ACCT_BAL_t table 2108 contains a list of all account balance elements that are contained in the various accounts. ACCT_BAL_t table 2108 includes an Account ID column that identifies the account, a Balance Element ID column that identifies the balance element type, a current balance column and a credit limit column.

An ACCT_PROD_t table 2110 contains a list of all of the products that are contained in the various accounts. An ACCT_PROD_t table 2110 contains an Account ID column that identifies the account, An element ID column that provides an index for the ACCT_PROD_t table, and a Product ID column that identifies the product that is included in the account identified by the Account ID.

A RATE_BAL_IMPACT_t table 2112 contains a list that describes the way that each rate will impact the various balances that are in an account. RATE_BAL_IMPACT_t table 2112 includes a rate ID column that identifies the rate, an Element ID column that provides an index for the RATE_BAL_IMPACT_t table, a Balance Element ID column that determines the balance element that is impacted, a fixed operand column that determines a minimum amount of the balance that is charged, a scaled operand column that determines the rate that the balance is charged after a free quantity, a free quantity column that determines the quantity that is provided for the fixed operand amount, and a type column that is used to describe the type of charge that is made (e.g. a debit, credit, or rebate.)

In one embodiment, the fixed operand, scaled operand, and free quantity are used according to the following cost formula:

cost = fixed operand + scaled operand * (quantity - free quantity).

It should be noted that in other embodiments, other formulas are used and other columns are provided in the RATE_BAL_IMPACT_t table to provide the operands required. It should also be generally noted that the columns in the tables shown are not intended to be all of the columns in each table, and shown for the purpose of illustrating the present example.

Figure 22 is a process flow diagram illustrating the process that runs when a billing event occurs for an account. The process starts at 2200. In a step 2202, a billing event is received by the system. The event includes a rate name, a quantity used, attributes that describe the event, and the account that is to be billed. Next, in a step 2204, the ACCOUNT_PROD_t is searched to determine all of the products that are owned by the account. In a step 2206, the RATE_t table is searched to retrieve all of the rates that match the rate name and the found products for the account. Thus, only rates that have a rate name that corresponds to the rate name of the event and that are associated with a product that is owned in the account are retrieved.

In a step 2208, all rates that do not have attributes that match the attributes of the billing event are discarded. What is left is a list of usable rates. The usable rates are then ordered by descending priority to obtain a rate stack in a step 2210. Next, in a step 2212, the quantity billed is allocated to each rate in order of priority and the applicable resources are adjusted according to the cost that is derived from the RATE_BAL_IMPACT_t table as described above. Each rate is allocated the maximum quantity that can be applies without exceeding a credit limit. If all of the rates are exhausted before the entire quantity billed is allocated, then control is transferred to a step 2214 and highest priority rate in the stack that has a default flag that indicates that the rate is an eligible default rate is used to allocate the remainder of the quantity. The process then ends at 2216. If the entire quantity billed is allocated in step 2214, the process ends at 2216 directly.

The above described process, combined with the tables shown in Figure 21 provides an extremely adaptable rating and billing system. In particular, the RATE_BAL_IMPACT_t table enables a single rate to impact multiple resources or even the same resource independently. For example, if the billing event is one hour of Type A connect time, then the customer can be charged hours, dollars, or any other resource, and can also be given free time, frequent flier miles or any other resource. In fact the customer could be billed dollars and rebated dollars. The cost formula shown above also allows the user to be billed a flat fee, an hourly rate, or some combination of a flat fee. Different rate brackets (similar to income tax brackets) applicable to different quantities can also be defined. Rates are prioritized and charged according to available credit limits and a default rate is applied when to available rate can be charged without violating a credit limit. When a credit limit is exceeded, service can be denied or a message sent.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

Figure 23 illustrates some of the tables stored in an RDBMS for the purpose of storing real time account balance information. A relational database table (Account_T) 2301 is used to store general account information. Table 2301 includes an Account_Id column that identifies the account. A Name column identifies the name of the person that the account belongs to and a Next_Bill_Time column lists the time that the account is to be billed next. The use of the "next bill time" in connection with calculating billing at that end of a billing cycle and with a rating operation will be discussed in further detail below. Other fields describing the account are typically included in Account_T as well, and the Name field is shown only for the purpose of an example.

A relational database table (Account_Balance_T) 2303 contains information about various account balances. An Account_Id column identifies the account and may be utilized by a RDBMS to join the Account_T and Account_Balance_T tables. An Element_Id column identifies the balance element. A Current _Balance column contains the current balance for each account balance element and a Credit_Limit column stores the credit limit for each account balance element.

Figure 24 is a process flow diagram illustrating a regular billing process implemented on a computer system that accesses the account balance information in an RDBMS to generate billing for a billing cycle.. The process runs sequentially on accounts.

At step 2401, an account being processed is locked. Locking the account creates a delay that prevents any of the account balances from being charged as a result of a billing event that occurs while the account is locked. In one embodiment locking the account is accomplished by setting a flag in the Account_T table. Next, at step 2403, the Next_Bill_Time column in the Account_T table is checked and determined if it is in the past at step 2405. Determining if the next bill time is in the past may include comparing the current time to the next bill time stored in the Account_T table.

If the next bill time is in the past then the system performs a close billing of the account at step 2407. Close billing of the account bills the account for any amounts due on the account during the billing cycle and it includes updating or advancing the next bill time in the Account_T table to the next billing cycle. Thus, close billing generates a bill for the account for the billing cycle.

The system unlocks the account at step 2409. If, at step 2405, it is determined that the next bill time is not in the past, then that is because the billing has already been performed and the next bill time was updated to a future time. In that case, control is transferred directly from step 2405 to step 2409.

The process of billing accounts typically operates sequentially on the accounts so at step 2411 it is determined if there is another account that needs to be processed. If there is, control returns to step 2401 and the next account is processed.

A rating operation determines a rate for the billing event and increments the various account balances impacted by the rate. When a transaction event that is billing occurs while accounts are being billed, it is possible that the account impacted by the billing event could be changed before it is closed out and billed. This would prevent a clean accounting close from being achieved. It is also possible that an account could have a billing event occur and have a rating operation occurring at the same time as a bill is being generated for the account. Such an occurrence might produce an unpredictable result. To prevent both of these problems, a special rating operation procedure is implemented. Whenever a billing event occurs, close billing is performed for the account impacted by the event if close billing has not already been performed. This so called event driven billing occurs out of sequence with respect to regular billing.

Figure 25 is a process flow diagram illustrating the process that occurs when a billing event occurs while the accounts are being billed. At step 2501, the system receives a transaction or billing event during the billing process. The account impacted by the billing event is locked at step 2503. Locking the account delays the regular billing process shown in Figure 24 from running on the account while the account is locked. Next, at step 2505, the next bill time is checked for the account to determine if it is in the past at step 2507. If the next bill time is in the past, then close billing is performed for the account and the next bill time is updated to the time (or date) of the next billing cycle at step 2509.

At step 2511, the balance impact(s) of the billing event are calculated and the balance elements of the account are updated. Thus, the billing event is posted to the account. If the next bill time is not in the past, then control is transferred directly from step 2507 to step 2511. At step 2513, the account is unlocked.

Thus, when a billing event occurs for an account occurs during a period when close billing is being run for the accounts, the next bill time is checked to determine whether billing has yet occurred for the account. If a bill has not yet been generated for the account, then billing is run immediately before the account balances are incremented. The interlocking feature between the processes shown in Figures 29 and 30 prevents regular billing from running on the account when event driven billing is running and vice versa.

The next bill time is analyzed to see if it is in the past in order to determine if an account has already been processed or billed during a close billing. Other mechanisms for determining whether an account has been billed during a close billing may be utilized. For example, a record

of the billing cycle where an account was last billed may also be utilized. Accordingly, the invention is not limited to the preferred embodiment described above.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

CLAIMS

1. In a computer system, a method of storing objects in a relational database, comprising the steps of:

defining a class of objects that are to be stored in the relational database, the objects of the class having at least one data member;

mapping each data member to at least one column in at least one relational database table; and

creating the at least one relational database table to store the objects of the class.

2. The method of claim 1, wherein the defining step comprises the step of generating a hierarchical tree that represents a definition of the data members of the class of objects.

3. The method of claim 1, wherein the hierarchical tree stores a hierarchy of classes including the class.

4. The method of claim 1, wherein the creating step comprises the steps of:

generating SQL statements for creating the at least one relational database table; and

sending the SQL statements to a relational database management system.

5. The method of claim 1, further comprising the step of instantiating an object of the class by instantiating a container object to store data members of the object.

6. The method of claim 5, wherein the container object is created utilizing a hierarchical tree that represents a definition of the data members of the class of objects.

7. The method of claim 1, further comprising the step of defining a subclass of objects that are to be stored in the relational database, the objects of the subclass inheriting the data members of the class of objects and having at least one additional data member.

8. The method of claim 7, further comprising the step of generating a hierarchical tree that comprises a definition of the at least one additional data member.

9. The method of claim 7, further comprising the steps of:

generating SQL statements for creating an additional relational database table to store the at least one additional data member; and

sending the SQL statements to a relational database management system.

10. The method of claim 1, wherein the at least one data member of the objects of the class comprise an array.

11. The method of claim 1, wherein the defining step comprises the step of receiving input of characteristics of data members.

12. The method of claim 1, wherein a characteristic indicates the data member is mandatory, optional, or read only.

13. The method of claim 1, wherein a characteristic indicates a default value for the data member.

14. In a computer system, a method of querying a relational database, comprising the steps of:

receiving a first query that is object-oriented and specifies information about objects of interest;

instantiating a query container object that comprises a query template based on the first query, an array for any arguments in the template query, and an array for any results in the template query;

utilizing the query container object, translating the first query into a second query in a relational database query language for accessing the specified information about the objects of interest that are stored by a relational database management system;

sending the second query to the relational database management system; and

receiving the specified information about the objects of interest from the relational database management system.

15. The method of claim 14, further comprising the step of utilizing a hierarchical tree to map data members of objects of interest to columns in the relational database.

16. The method of claim 14, further comprising the step of instantiating a results container object to store the specified information about the objects of interest.

17. In a computer system, a method of deferring allocation of storage for array elements of objects comprising the steps of:

receiving a request to instantiate an object of a class where the class has a definition that specifies a default value for each data member of an array element;

allocating storage space for the object without storage space for an array element if the instantiation request does not specify an initial value for any of the data members of the array element;

receiving a request to modify a data member of the array element;

determining if storage space for the array element has been allocated;

if storage space for the array element has not been allocated, allocating storage space for the array element and initializing each data member of the array element to the specified default value; and

modifying the data member of the array element as specified in the modification request.

18. The method of claim 17, wherein each array element is stored as a row in a relational database table.

19. A computer system for storing objects in a relational database, comprising:

an object-oriented application for receiving a definition of a class of objects that are to be stored in the relational database, the objects being stored in a temporary storage;

a memory for the temporary storage of the objects;

an object server for retrieving the objects from the memory and issuing statements in a relational database query language to store data of the objects; and

a relational database management system for receiving the statements and storing the data of the objects in persistent storage as relational database tables.

20. The system of claim 19, further comprising a hierarchical tree that represents the definition of the class of objects including data members.

21. The system of claim 20, wherein the hierarchical tree stores a mapping from each data member of the class of objects to at least one column in the relational database tables.

22. The system of claim 20, wherein the hierarchical tree stores a hierarchy of classes including the class.

23. A method of assigning payment resources to an account balance stored in an RDBMS table comprising:

instantiating a balance adjusting container object that includes an account element identification number, a balance element identification number of a balance element that will be adjusted and an adjusting amount;

formulating an RDBMS query to determine whether an entry in an account balance table exists that corresponds to the account element identification number and the balance element identification number;

if an entry in the account balance table is found, then adjusting the found entry in the account balance table by the adjusting amount;

if no entry in the account balance table is found, then creating an entry in the account balance table with a default balance and incrementing the created entry by the adjusting amount.

24. A method as recited in claim 23 wherein the default balance is zero.

25. A method as recited in claim 23 wherein the RDBMS query is an SQL query.

26. A method as recited in claim 23 further including sending the balance adjusting container to and object server and wherein the object server formulates the RDBMS query.

27. A method as recited in claim 23 wherein the object server formulates the RDBMS query using an hierarchical tree.

28. A method of defining a payment resource for inclusion in a real time transaction processing system comprising:

instantiating a balance element creating container object;

including data fields in the balance element creating container object including a balance element identification number field and providing a balance element identification number in the balance element identification number field.

adding a new row to a payment resources RDBMS table using the information from the balance element creating container object.

29. A method as recited in claim 28 wherein an object server determines the new row that is to be added in the payment resources RDBMS table.

30. A method as recited in claim 29 wherein the object server checks the balance element creating container object to make certain that the balance element creating container contains all of a set of required fields.

31. A method as recited in claim 28 wherein including data fields in the balance element creating container object further includes including a resource name field and a rounding amount field in the balance element creating container object.

32. A method of determining a rate for adjusting a real time balance including:

receiving a billing event, the billing event including a rate name, a quantity used, an attribute that describes the event, and an account to be billed;

searching a product table to determine all of the products in the account to be billed;

searching a rate table to determine all of the rates that correspond to the rate name and the products in the account to be billed;

eliminating rates that do not match the attribute that describes the event;

ordering the remaining rates by descending priority to obtain a rate stack; and

for each rate in descending order of priority, allocating the maximum portion of the quantity used that can be applied using the rate without violating a credit limit.

33. A method as recited in claim 32 further including applying a default rate to a portion of the quantity used that cannot be applied using one of the remaining rates without violating a credit limit.

34. In a computer system, a method of performing real time billing of accounts, comprising the steps of:

during close billing of a billing cycle, receiving a transaction event for an account;

locking the account;

determining if the account has already been billed in the billing cycle;

if the account has not already been billed in the billing cycle, close billing the account without billing the transaction event;

posting the transaction event to the account; and

unlocking the account.

35. The method of claim 34, wherein the determining step includes the step of determining if the next bill time is in the past.

36. The method of claim 35, wherein the billing step includes the step of advancing the next bill time of the account to a next billing cycle.

37. The method of claim 35, wherein the next bill time is compared to a current time.

38. The method of claim 34, wherein the transaction event is a billing event.

39. In a computer system, a method of performing real time billing of accounts, comprising the steps of:

during close billing of a billing cycle, receiving a billing event for an account;

locking the account;

determining if the account has already been billed in the billing cycle by checking if a next bill time of the account is in the past;

if the account has not already been billed in the billing cycle, close billing the account without billing the billing event;

posting the billing event to the account; and

unlocking the account.

40. The method of claim 39, wherein the billing step includes the step of advancing the next bill time of the account to a next billing cycle.

41. The method of claim 39, wherein the next bill time is compared to a current time.

42. A computer implemented real time billing system, comprising:

a processor;

a memory coupled to the processor that stores accounts;

a billing process operating on the processor that bills accounts at the end of a billing cycle and upon receiving a transaction event for an account, the billing process locks the account, close bills the account without billing the transaction event if the account has not already been billed in the billing cycle, posts the transaction event to the account, and unlocks the account.

43. The system of claim 42, wherein it is determined if the account has not already been billed in the billing cycle if a next bill time is in the past.

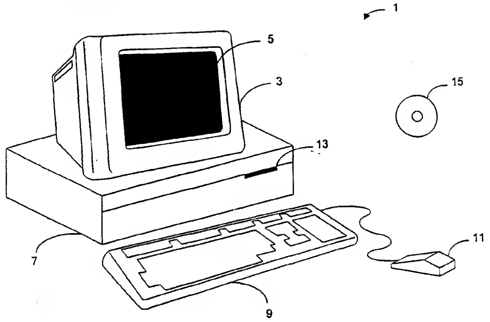

1
5
3
15
13
11
7
9

FIG. 1

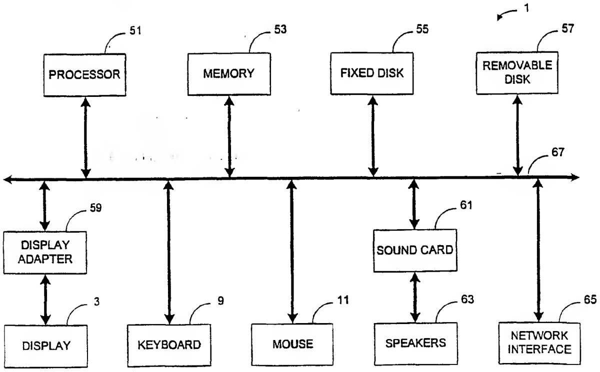

1
51
53
55
57
PROCESSOR
MEMORY
FIXED DISK
REMOVABLE DISK
67
59
61
DISPLAY ADAPTER
SOUND CARD
3
9
11
63
65
DISPLAY
KEYBOARD
MOUSE
SPEAKERS
NETWORK INTERFACE

FIG. 2

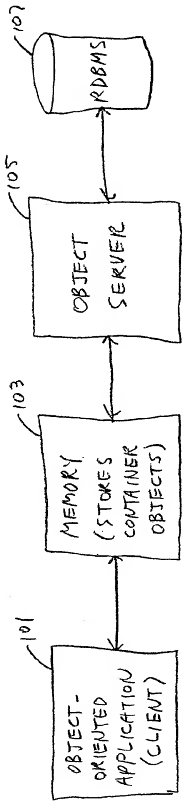

102
RDBMS
105
OBJECT SERVER
103
MEMORY (STORES CONTAINER OBJECTS)
101
OBJECT-ORIENTED APPLICATION (CLIENT)

FIG. 3

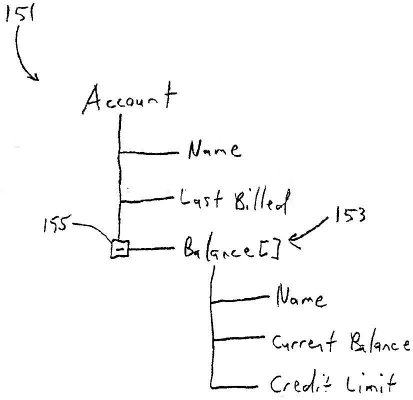

151
Account
Name
Last Billed
155
Balance[]
153
Name
Current Balance
Credit Limit

FIG. 4

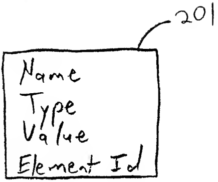


FIG. 5A

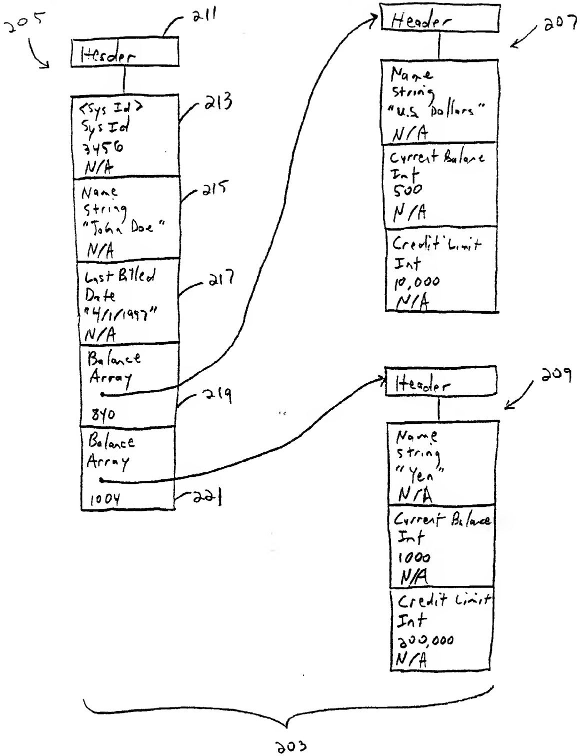

205
211
Header
<Sys Id>
Sys Id
3456
N/A
213
Name
String
"John Doe"
N/A
215
Last Billed
Date
"4/1/1997"
N/A
217
Balance
Array
840
219
Balance
Array
1004
221
Header
207
Name
String
"U.S. Dollars"
N/A
Current Balance
Int
500
N/A
Credit Limit
Int
10,000
N/A
Header
209
Name
String
"Yen"
N/A
Current Balance
Int
1000
N/A
Credit Limit
Int
200,000
N/A
203

FIG. 5B

Account_T 251

Id	Name	Last_Billed
3456	John Doe	"4/1/1997"
⋮		

Account_Balance_T 253

Id	Element_Id	Name	Current_Balance	Credit_Limit
3456	840	U.S. Dollars	500	10,000
3456	1004	Yen	1000	200,000
⋮				

FIG. 6

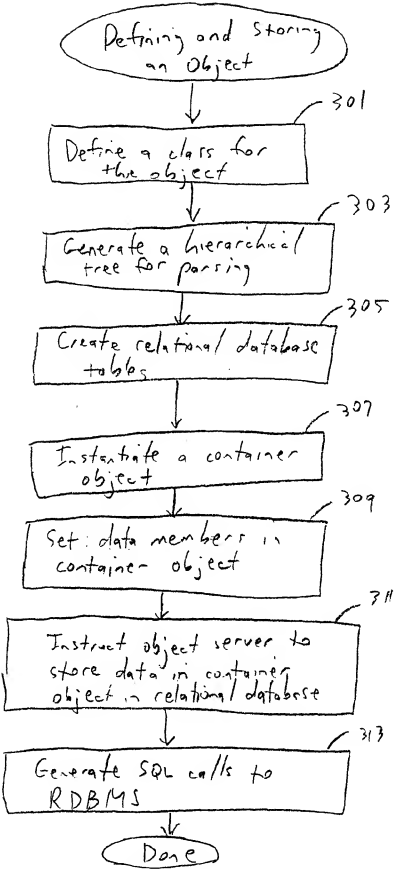

Defining and Storing an Object
301
Define a class for the object
303
Generate a hierarchical tree for parsing
305
Create relational database tables
307
Instantiate a container object
309
Set data members in container object
311
Instruct object server to store data in container object in relational database
313
Generate SQL calls to RDBMS
Done

FIG. 7

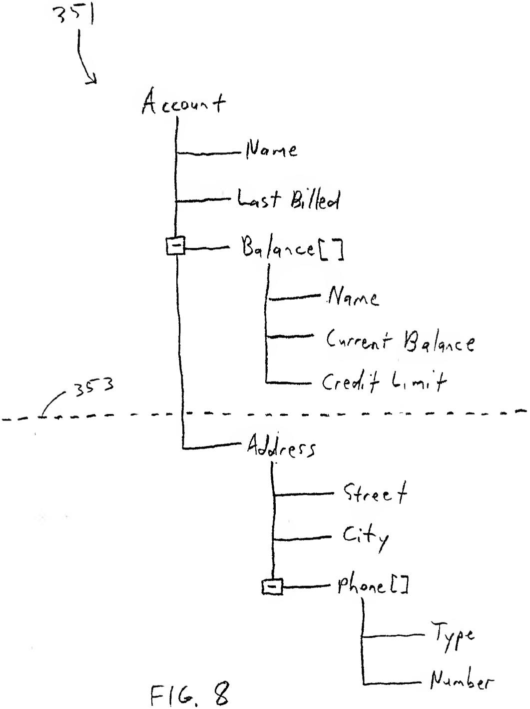

351
Account
Name
Last Billed
Balance[]
Name
Current Balance
Credit Limit
353
Address
Street
City
phone[]
Type
Number

FIG. 8

Account_Address_T

401

Id	Street	City
3456	Elm St.	San Jose
⋮		

Account_Address_Phone_T

403

Id	Element_Id	Type	Number
3456	56	Home	555-1111
3456	58	Work	555-1112
⋮			

FIG. 9

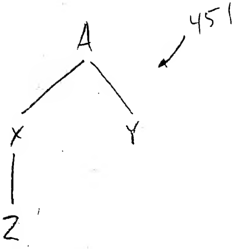


FIG. 10

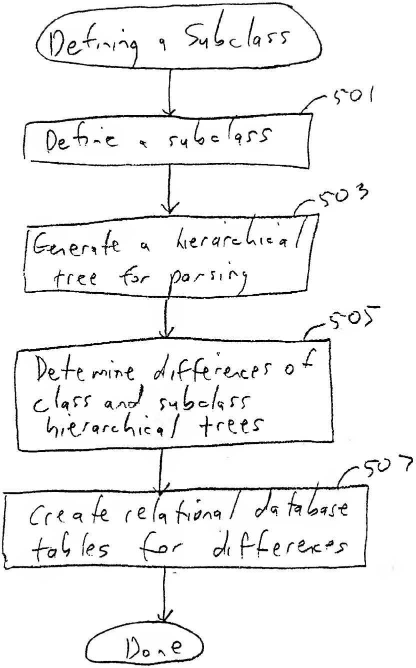

Defining a Subclass
501
Define a subclass
503
Generate a hierarchical tree for parsing
505
Determine differences of class and subclass hierarchical trees
507
Create relational database tables for differences
Done

FIG. 11

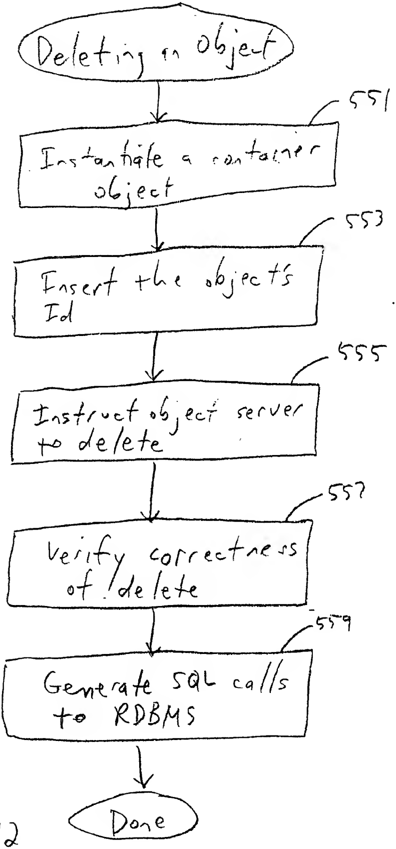

Deleting an Object
551
Instantiate a container object
553
Insert the object's Id
555
Instruct object server to delete
557
Verify correctness of delete
559
Generate SQL calls to RDBMS
Done

FIG. 12

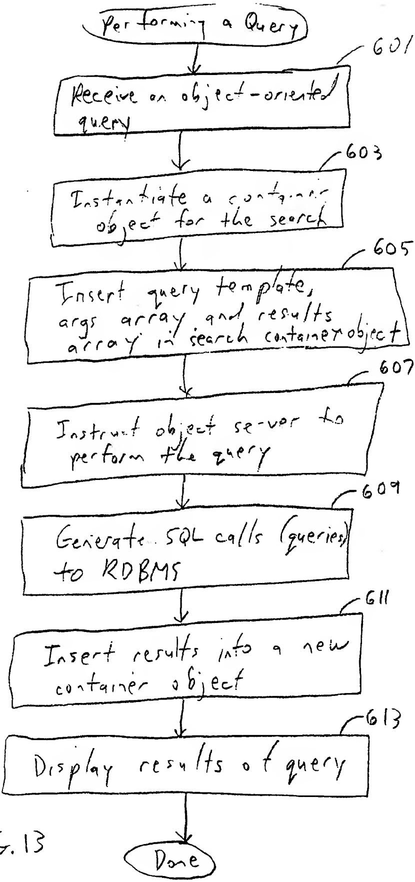

Performing a Query
601
Receive an object-oriented query
603
Instantiate a container object for the search
605
Insert query template, args array and results array in search container object
607
Instruct object server to perform the query
609
Generate SQL calls (queries) to RDBMS
611
Insert results into a new container object
613
Display results of query
Done

FIG. 13

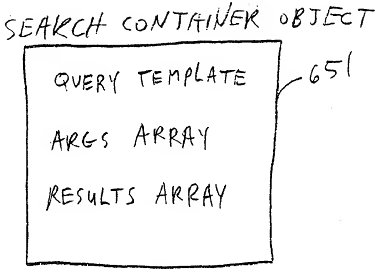

SEARCH CONTAINER OBJECT
QUERY TEMPLATE
ARGS ARRAY
RESULTS ARRAY
651

FIG. 14

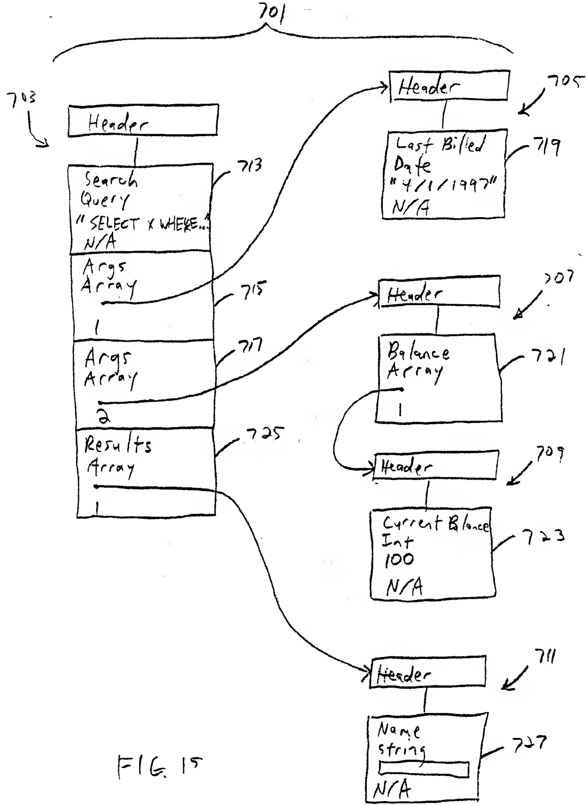

701
703
Header
Search
Query
"SELECT X WHERE..."
N/A
713
Args
Array
1
715
Args
Array
2
717
Results
Array
1
725
Header
705
Last Billed
Date
"4/1/1997"
N/A
719
Header
707
Balance
Array
1
721
Header
709
Current Balance
Int
100
N/A
723
Header
711
Name
string
N/A
727

FIG. 15

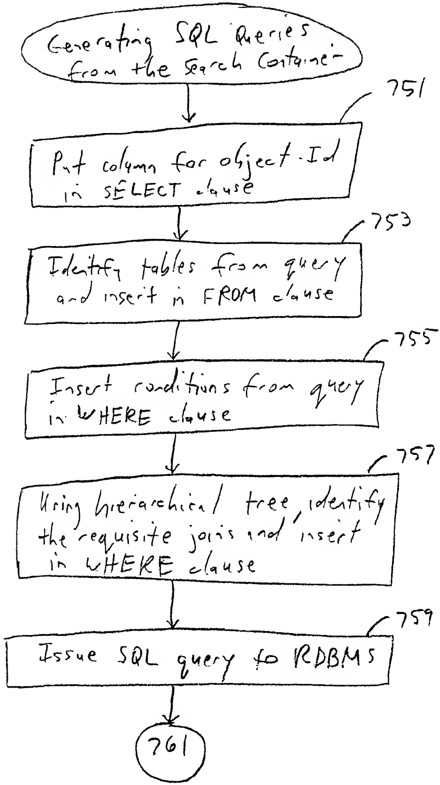

Generating SQL Queries from the Search Container
751
Put column for object Id in SELECT clause
753
Identify tables from query and insert in FROM clause
755
Insert conditions from query in WHERE clause
757
Using hierarchical tree, identify the requisite joins and insert in WHERE clause
759
Issue SQL query to RDBMS
761

FIG. 16A

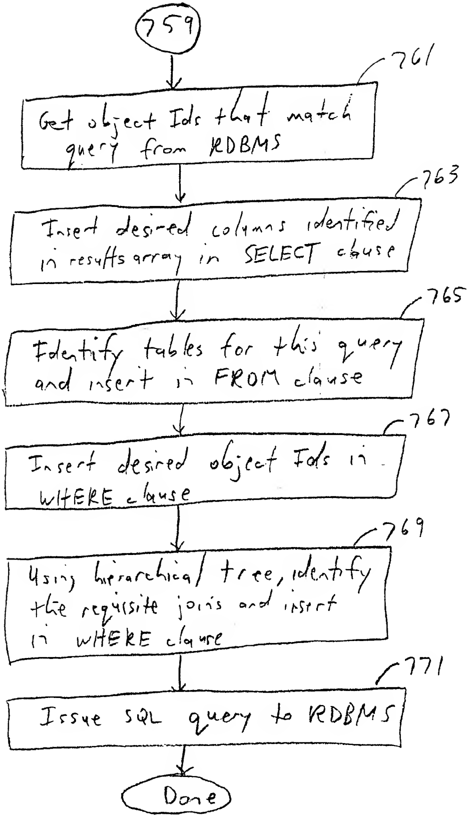

759
761
Get object Ids that match query from RDBMS
763
Insert desired columns identified in results array in SELECT clause
765
Identify tables for this query and insert in FROM clause
767
Insert desired object Ids in WHERE clause
769
Using hierarchical tree, identify the requisite joins and insert in WHERE clause
771
Issue SQL query to RDBMS
Done

FIG. 16B

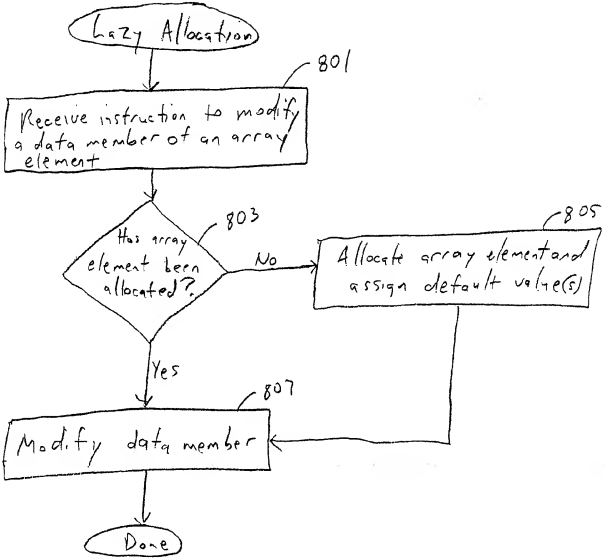

Lazy Allocation
801
Receive instruction to modify a data member of an array element
803
Has array element been allocated?
No
805
Allocate array element and assign default value(s)
Yes
807
Modify data member
Done

FIG. 17

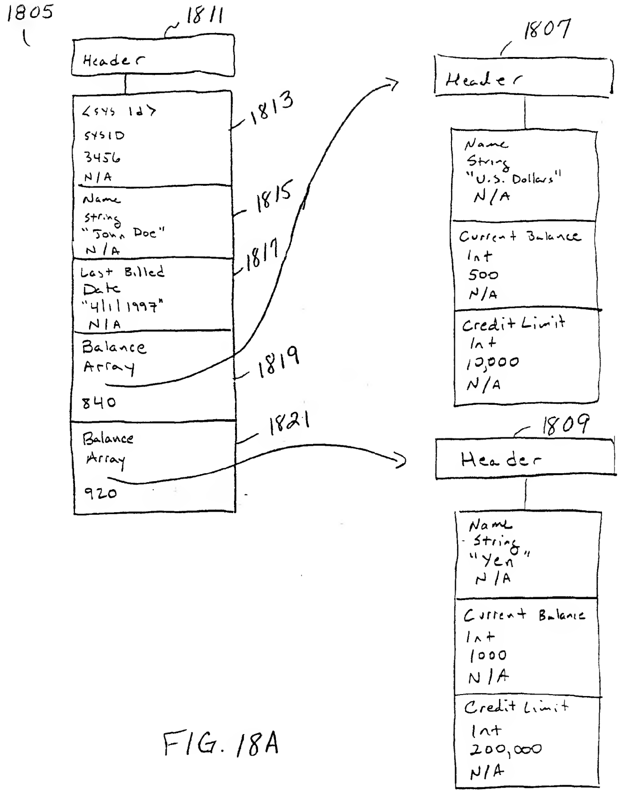

1805
1811
Header
<SYS Id>
SYSID
3456
N/A
1813
Name
String
"John Doe"
N/A
1815
Last Billed
Date
"4/1/1997"
N/A
1817
Balance
Array
840
1819
Balance
Array
920
1821
1807
Header
Name
String
"U.S. Dollars"
N/A
Current Balance
Int
500
N/A
Credit Limit
Int
10,000
N/A
1809
Header
Name
String
"Yen"
N/A
Current Balance
Int
1000
N/A
Credit Limit
Int
200,000
N/A

FIG. 18A

Account_T 1851

Sys_Id	Id	Name	Last_Billed
<Sys Id>	3456	John Doe	"4/1/1997"
•			
•			
•			

Account_Balance_T 1853

Id	Element_Id	Current_Balence	Credit_Limit
3456	840	500	10,000
3456	1004	1000	200,000
•			
•			
•			

Resources_T 1854

Balance Element Id	Resource Name	GUI Name	Rounding
840	U.S. dollars		.01
920	Yen		.01
1,000,001	Frequent Flyer Miles		1.

Figure 18B

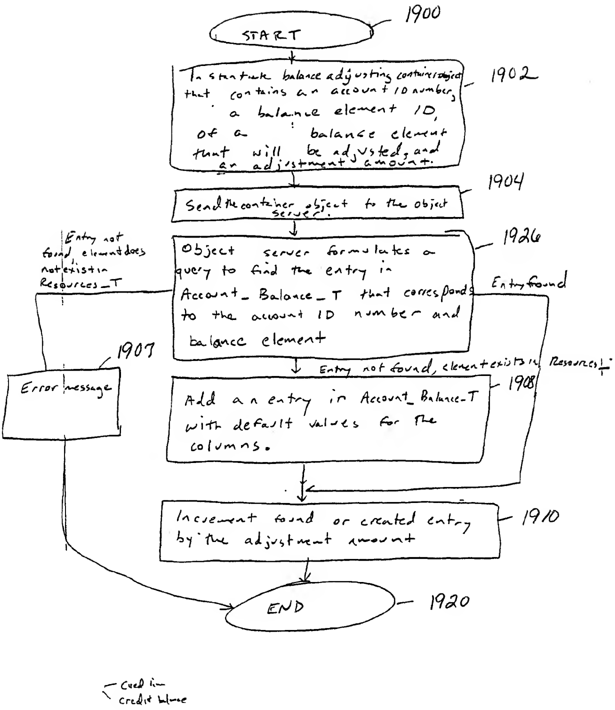

1900
START
1902
Instantiate balance adjusting container object that contains an account ID number, a balance element ID, of a balance element that will be adjusted, and an adjustment amount.
1904
Send the container object to the object server.
1926
Object server formulates a query to find the entry in Account_Balance_T that corresponds to the account ID number and balance element
Entry not found, element does not exist in Resources_T
Entry found
1907
Error message
Entry not found, element exists in Resources_T
1908
Add an entry in Account_Balance_T with default values for the columns.
1910
Increment found or created entry by the adjustment amount
END
1920
Cred lim
credit balance

FIG. 19

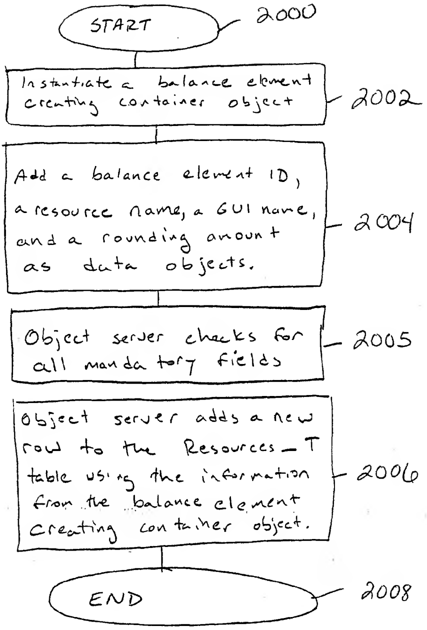

START
2000
Instantiate a balance element creating container object
2002
Add a balance element ID, a resource name, a GUI name, and a rounding amount as data objects.
2004
Object server checks for all mandatory fields
2005
Object server adds a new row to the Resources_T table using the information from the balance element creating container object.
2006
END
2008

FIG. 20

Product_t

2102

PRODUCT ID	NAME	ATTRIBUTES
6492	PRODUCT A PRODUCT B PRODUCT C	

RATE_t

2104

RATE ID	PRODUCT-ID	NAME	PRIORITY	DEFAULT FLAG	ATTRIBUTES
1234	6492 6492 6492	TYPE1, TYPE2		1	

ACCT_t

2106

ACCT-ID	<ACCOUNT ATTRIBUTES>

ACCT_BAL_t

2108

ACCOUNT ID	ELEMENT ID	CURR BAL	CREDIT LIMIT

ACCT_PROD_t

ACCOUNT ID	ELEMENT ID	PRODUCT ID

2110

RATE_BAL_IMPACT_t

2112

RATE ID	ELEMENT ID	BAL_ELEMENT	FIXED OPERAND	SCALED OPERAND	FREE QUANTITY	TYPE
1234 1234 1234	1 2 3	840 1004 840				

FIG 21

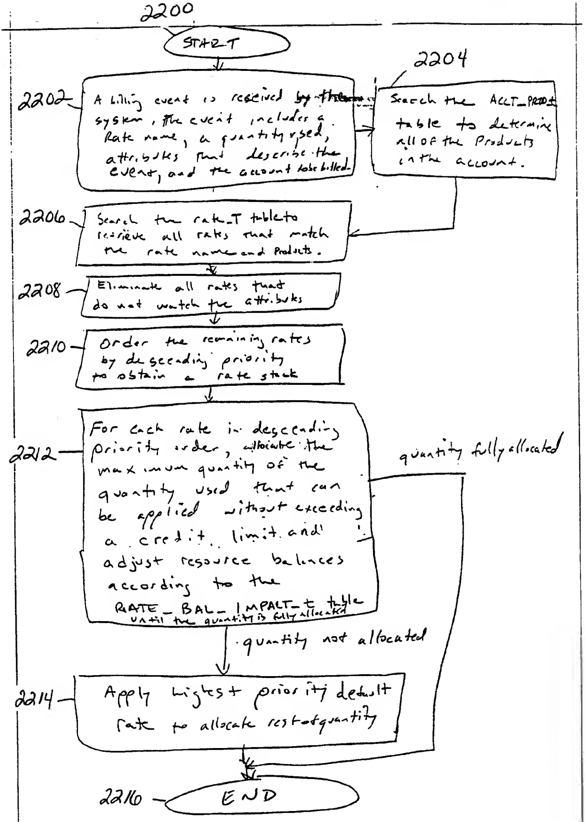

2200
START
2202
A billing event is received by the system, the event includes a Rate name, a quantity used, attributes that describe the event, and the account to be billed.
2204
Search the ACCT_PROD_t table to determine all of the Products in the account.
2206
Search the rate_T table to retrieve all rates that match the rate name and Products.
2208
Eliminate all rates that do not match the attributes
2210
Order the remaining rates by descending priority to obtain a rate stack
2212
For each rate in descending priority order, allocate the maximum quantity of the quantity used that can be applied without exceeding a credit limit and adjust resource balances according to the RATE_BAL_IMPACT_t table until the quantity is fully allocated
quantity fully allocated
quantity not allocated
2214
Apply highest priority default rate to allocate rest of quantity
2216
END

FIG. 22

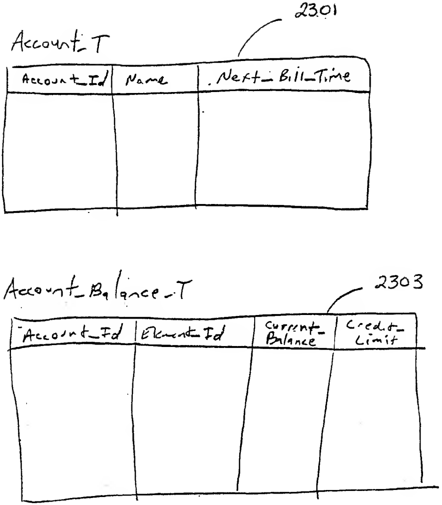

2301
Account_T
Account_Id
Name
Next_Bill_Time
Account_Balance_T
2303
Account_Id
Element_Id
Current_Balance
Credit_Limit

Fig. 23

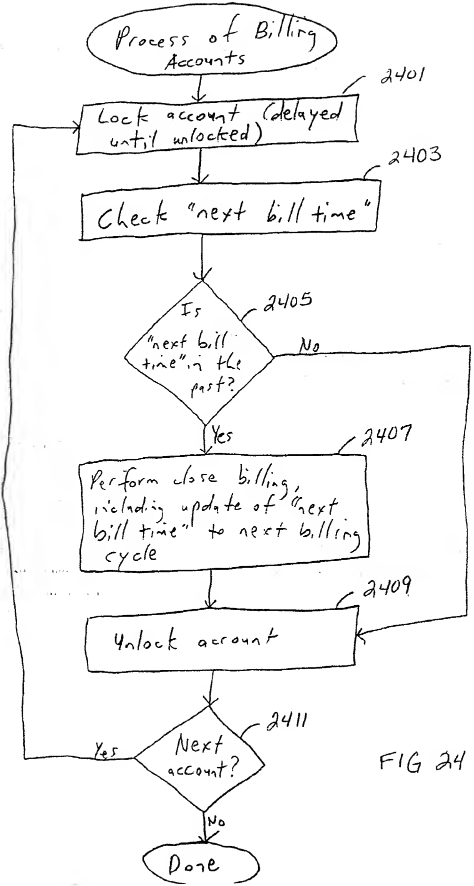

Process of Billing Accounts
2401
Lock account (delayed until unlocked)
2403
Check "next bill time"
2405
Is "next bill time" in the past?
No
Yes
2407
Perform close billing, including update of "next bill time" to next billing cycle
2409
Unlock account
2411
Next account?
Yes
No
Done

FIG 24

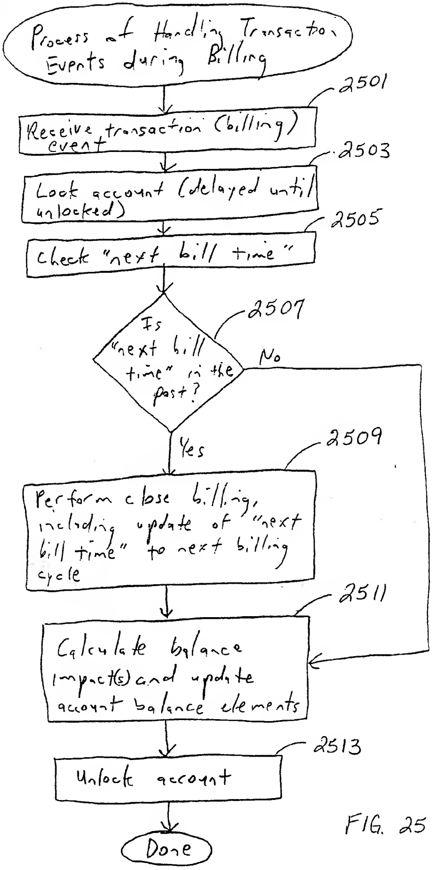

Process of Handling Transaction Events during Billing
2501
Receive transaction (billing) event
2503
Lock account (delayed until unlocked)
2505
Check "next bill time"
2507
Is "next bill time" in the past?
No
Yes
2509
Perform close billing, including update of "next bill time" to next billing cycle
2511
Calculate balance impact(s) and update account balance elements
2513
Unlock account
Done

FIG. 25

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/10116

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 96 34350 A (ASPECT DEV INC) 31 October 1996 see page 4, line 12 - page 9, line 20 see page 23, line 20 - page 25, line 18 see page 31, line 18 - page 31, line 22 see page 47, line 7 - page 47, line 8 see figures 1,2	1-22
Y		23-43

X	WO 97 03406 A (WALL DATA INC ;KAWAI KENJI (US)) 30 January 1997 see abstract see page 23, line 28 - page 24, line 7	1,4, 19-22

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 October 1998	21/10/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Abbing, R

2

INTERNATIONAL SEARCH REPORT

Inte onal Application No
PCT/US 98/10116

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	CHENHO KUNG: "OBJECT SUBCLASS HIERARCHY IN SQL: A SIMPLE APPROACH" COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, vol. 33, no. 7, 1 July 1990, pages 117-125, XP000143088 see the whole document	1,7-9
A	US 5 530 853 A (SCHELL DAVID J ET AL) 25 June 1996 see column 1, line 13 - column 2, line 10 see claims	1,5,6, 14-16
Y	WO 95 27255 A (REGAN TIMOTHY ;WILSON JEREMY (GB); FREESTONE DAVID (GB); BRITISH T) 12 October 1995 see the whole document	23-43
A		1-22
A	WO 95 04960 A (PERSISTENCE SOFTWARE INC) 16 February 1995 see abstract see page 5, line 33 - page 7, line 2	19-22

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 98/10116

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9634350 A	31-10-1996	EP 0823092 A	11-02-1998
WO 9703406 A	30-01-1997	US 5717924 A	10-02-1998
		AU 6251796 A	10-02-1997
		CA 2222583 A	30-01-1997
		EP 0838060 A	29-04-1998
		NO 980068 A	06-01-1998
US 5530853 A	25-06-1996	JP 6202918 A	22-07-1994
		JP 8020982 B	04-03-1996
WO 9527255 A	12-10-1995	SG 43130 A	17-10-1997
		AU 2080095 A	23-10-1995
		CA 2186626 A	12-10-1995
		EP 0753179 A	15-01-1997
WO 9504960 A	16-02-1995	US 5615362 A	25-03-1997
		US 5706506 A	06-01-1998

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30, 17/60 — A1

(11) International Publication Number: WO 98/52131

(43) International Publication Date: 19 November 1998 (19.11.98)

(21) International Application Number: PCT/US98/10116

(22) International Filing Date: 13 May 1998 (13.05.98)

(30) Priority Data:

08/856,375	14 May 1997 (14.05.97)	US
08/856,313	14 May 1997 (14.05.97)	US
08/856,372	14 May 1997 (14.05.97)	US

(71) Applicant: PORTAL INFORMATION NETWORK [-/US]; Suite 200, 20863 Stevens Creek Boulevard, Cupertino, CA 95014 (US).

(72) Inventors: OWENS, Gary, L.; 565 Hope Street, Mountain View, CA 94041 (US). LABUDA, David, S.; 213 Correas Street, Half Moon Bay, CA 94019 (US).

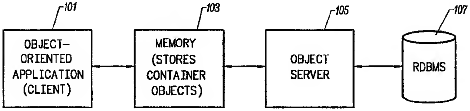
(74) Agent: VAN PELT, Lee; Beyer & Weaver, LLP, P.O. Box 61059, Palo Alto, CA 94306 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR OBJECT ORIENTED STORAGE AND RETRIEVAL OF DATA FROM A RELATIONAL DATABASE TO IMPLEMENT A REAL TIME BILLING SYSTEM



(57) Abstract

Systems and methods for accessing a relational database through an object-oriented querying interface are provided. A class of objects that are to be stored in the relational database are defined. One or more relational database tables are created and a mapping is produced that maps each data member of an object to one or more columns in a relational database table. Additionally, object-oriented paradigms like inheritance may be supported and the allocation of storage for array elements may be deferred until necessary. A container object that allows a user to define new payment resources without requiring the user to redesign a relational database system used for persistent storage of transaction informatiion is also disclosed. A real time billing system for accounts that locks out transaction events when the billing process is underway is also disclosed. When a transaction event is received that should be posted to an account during the billing process, the account is locked.

METHOD AND APPARATUS FOR OBJECT ORIENTED STORAGE AND RETRIEVAL OF DATA FROM A RELATIONAL DATABASE TO IMPLEMENT A REAL TIME BILLING SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to methods and apparatuses for transferring data to and from a first memory that is organized according to an object-oriented scheme to a second memory that is organized according to a relational database management scheme.

In one embodiment, the invention relates to transferring data to and from a transient storage that is organized according to an object-oriented scheme to a persistent storage that is organized according to a relational database management scheme. In certain embodiments, the relational database in persistent storage is designed by an object server. This includes defining the tables of the relational database as well as the various columns. The object server then stores and retrieves data from the various tables defined in persistent storage according to a hierarchical tree that maps data encapsulated within objects to table locations in the relational database found in persistent storage.

In another embodiment, the invention relates to debiting and crediting existing payment resources or creating new payment resources using an object oriented scheme. Data relating to payment resources is transferred to and from an object server using a container object. The object server interacts with both a transient storage that is organized according to an object-oriented scheme and a persistent storage that is organized according to a relational database management scheme. The relational database in persistent storage is designed by the object server. This includes defining the tables of the relational database as well as the various columns. The object server then stores and retrieves data from the various tables defined in persistent storage according to a hierarchical tree that maps data encapsulated within objects to table locations in the relational database found in persistent storage.

When an event occurs that requires payment from a payment source, or when authorization is sought for an event that will require payment from a payment source, a rating engine analyzes a number of rates that are available, resources that are available, and available credit limits to create a rate stack that determines authorization and charging for a transaction.

In another embodiment, the invention relates to preventing transactions that occur after the end of a billing period but before billing has been completed from being included in bills generated by a real time billing system.

2. Description of the Related Art

There are well known tradeoffs associated with relational database and object -oriented database designs. Relational databases are commonly optimized for fast, efficient searching. This is largely the result of the fact that relational databases are built from a set of tables that contain related columns. The tables are indexed in an efficient manner by the relational database so that searches may be performed in an optimal manner. While organizing information into a complex related set of tables helps speed searching, a thorough knowledge of the tables is required to specify data that is to be retrieved or to specify where data is to be stored. Furthermore, changing the structure of the tables to add a column may require extensive programming and rewriting of existing code. Another problem in many relational database management systems (RDBMSs) is that columns in tables that contain no information or are not used nevertheless take up space in memory.

A standard relational query language, Structured Query Language (SQL) is used to query most popular relational databases. SQL requires that the person who specifies a query know what tables and columns contain the information that is to be compared against the query. For example, in order to look for all customers in a city, the user must know both the name of the table that contains city information and also the name of the column in that table that contains the city information. It is also necessary that the user know the tables that should be joined to accomplish the search. Likewise, in order to store information in the proper column of the proper table, the user must know the name of the table and column in which the information should be stored.

In contrast, it is easier to query, modify and write information to object-oriented databases. Instead of specifying a table and column for storing or retrieving information, related data is encapsulated in an object. The object may be read into memory and all encapsulated data may be readily accessed. Searching, however, is not as efficient as relational database searching. Entire objects are read into memory in order to check the relevant encapsulated data members. Similarly, store operations are performed on entire objects. Thus, this methodology is not very well suited for on-line transaction processing (OLTP) where transaction rates are high but often only portions of the objects are desired.

Attempts to make object-oriented relational databases have for the most part merely added an object-oriented interpretation to a relational database structure. For example, rows in an existing relational database structure may be interpreted as an object, with each column representing an encapsulated data member. This arrangement, however, does not realize the full power of an object-oriented database. For example, inheritance is not supported so subclasses of objects may not be defined. Additionally, the problem of adding data members to objects is

not addressed. Still further, adding a column with no data still allocates large chunk of storage for that column, even if the column is never used.

In view of the foregoing, there is a need for methods and apparatuses for taking advantage of the programming, storage and querying ease of an object-oriented database while enjoying the searching speed of a relational database.

Billing has become increasingly complex in the field of on line transaction processing. In particular, convergent billing, where multiple services are billed to a customer on a single bill has become an important goal of on line transaction processing systems. In such systems, multiple sources of payment may be used to pay for services. For example, a phone company may provide both long distance phone service and internet access and may also handle billing for a certain on line services accessed by a customer. As the customer accesses one service, it may be desired to give the customer free time on one of the other services as an incentive to use both services. Additionally, it may be desired to give the customer some other sort of incentive such as frequent flier miles whenever the customer accesses a certain service. Payment for a given service may then be made from any one of these multiple resources, which must be tracked.

The problem becomes more complex when an authorization event or a payment event occurs. Different payment rates from different sources must be analyzed to determine the rate to be applied. In the case of an authorization, different credit limits applying to different sources must be checked.

One of the most important requirements of such a transaction processing system is the need to add payment resources and rate schemes without extensive reprogramming. Traditional payment systems have not been successful in meeting this requirement. The need for fast, optimized searching of a very large database has militated in favor of relational data base designs for the storage of customer, payment source, and rate information. Relational databases are built from a set of tables that contain related columns. The tables are indexed in an efficient manner by the relational database so that searches may be performed in an optimal manner. While organizing information into a complex related set of tables helps speed searching, a thorough knowledge of the tables is required to specify data that is to be retrieved or to specify where data is to be stored. Creating new data and creating new types of data such as new payment resources generally requires modifying the design of the tables to accommodate the new data and often requires extensive rewriting of existing the system program code. This means that a high level of expertise and familiarity with the database design is required to add payment resources.

Another problem in many relational database management systems (RDBMS's) is that columns in tables that contain no information or are not used nevertheless take up space in memory. Thus, when new payment resources are created. a large amount of memory may be

allocated for tracking the payment source for every customer. Old payment resources that are no longer used also may continue to take up storage space until the database is restructured to remove them.

A standard relational query language, Structured Query Language (SQL) is used to query most popular relational databases. SQL requires that the person who specifies a query know what tables and columns contain the information that is to be compared against the query. For example, in order to look for all customers in a city, the user must know both the name of the table that contains city information and also the name of the column in that table that contains the city information. It is also necessary that the user know the tables that should be joined to accomplish the search and how to join those tables. Likewise, in order to store information in the proper column of the proper table, the user must know the name of the table and column in which the information should be stored.

In contrast, it is easier to query, modify and write information to object-oriented databases. Instead of specifying a table and column for storing or retrieving information, related data is encapsulated in an object. The object may be read into memory and all encapsulated data may be readily accessed. Searching, however, is not as efficient as relational database searching. It would be desirable if an object oriented database could be used to add payment resources and rates and to retrieve information into a transient memory, and if a relational database could be used for persistent storage of data.

In view of the foregoing, there is a need for methods and apparatuses for defining payment resources using an object-oriented database associated with a transient memory and creating a relational data base in persistent memory that stores payment source information. It would also be useful if an efficient system for charging payments at different rates could be developed.

Real time event rating and billing systems offer important advantages over traditional billing systems. In a traditional billing system, transaction events are time stamped and entered into a database. In order to determine an account balance at a certain point in time it is necessary to search the database and retrieve all of the transaction events that occurred within the time period of interest and compute the account balance based on the transaction events found. A disadvantage of such a system is that real time balances are not kept, that is, a search and calculations are required to determine current balances at any point in time. No running balance total is stored in memory.

Real time event rating and billing systems, on the other hand, keep a running total of balances. It may be useful to have such a running total readily available in memory for several reasons. For example, it may be desirable to authorize transactions in real time based on account

balance or apply a rate to a transaction in real time as a function of certain existing balances. Generally, a real time rating system is useful whenever real time account balances are needed.

In most billing systems, bills are generated periodically. Typically, bills are generated for a specific period defined by an opening time and a closing time. It is generally desirable that the close of the billing period be precisely determined and that transaction events outside of the period be excluded from the billing. Generating a clean accounting close is relatively simple for a non-real time system. The definition of the search criteria provides a time cut off that retrieves only transaction events that occur before the close of a billing period. Generating a clean accounting close for a real time event rating and billing system which is continuously adjusting account balances as transaction events occur, on the other hand, presents a more difficult problem. The billing process is not instantaneous and generally takes a finite amount of time to run that in many applications is at least a couple of hours. During that time, if a transaction event occurs, then it may change an account balance before that balance is billed. The transaction would then be improperly recorded since it occurred after the close of the accounting period.

In view of the foregoing, there is a need for methods and apparatuses for providing a clean accounting close for a real time transaction processing system that keeps running balance totals for accounts and does not generate balances using time delimited search criteria.

SUMMARY OF THE INVENTION

Accordingly, the present invention provides an object server that maps data that is represented in transient memory according to an object-oriented scheme to data that is represented in persistent memory according to a relational database scheme. In certain embodiments, the object server generates appropriate tables and columns for a relational database scheme automatically so that an object-oriented scheme generated by a user may be efficiently stored and searched in persistent memory. Preferably, array elements are represented as rows in a table, not as columns so that storage space is not wasted with place holder data.

The present invention also provides a container object that allows a user to define new payment resources without requiring the user to redesign a relational database system used for persistent storage of transaction information. An object server maps data that is represented in transient memory according to an object-oriented scheme to data that is represented in persistent memory according to a relational database scheme. The object server generates appropriate tables and columns for a relational database scheme automatically so that the object-oriented scheme generated by a user may be efficiently stored and searched in persistent memory. Preferably, array elements are represented as rows in a table, not as columns so that storage space is not wasted with place holder data. In certain embodiments, a rating engine is provided that searches available rates and creates a rate stack for the purpose of authorizing transactions and adjusting payment source balances when authorization events or payment events occur.

The present invention also provides an interlock system that locks an account while the account is being billed and then releases the account. When a transaction event occurs during billing, that account is locked and billing for the account is generated immediately. The transaction event is then processed and the account balance is adjusted before the account is unlocked. Subsequent billing for that account during the same period is avoided by checking whether a "last billed" field for the account contains a time that is prior to the closing time for the billing period.

In one embodiment, the invention provides a computer implemented method of storing objects in a relational database comprising the steps of: defining a class of objects that are to be stored in the relational database, the objects of the class having at least one data member; creating at least one relational database table to store the objects of the class; and mapping each data member to at least one column in the at least one relational database table. Subclasses of objects may be defined that inherit the data members of a parent class. The data members for objects of the subclass are typically stored in additional relational database tables.

In another embodiment, the invention provides a computer implemented method of querying a relational database comprising the steps of: receiving a first query that is object-oriented and specifies information about objects of interest; instantiating a query container object that comprises a query template based on the first query, an array for any arguments in the template query, and an array for any results in the template query; utilizing the query container object, translating the first query into a second query in a relational database query language for accessing the specified information about the objects of interest that are stored by a relational database management system; sending the second query to the relational database management system; and receiving the specified information about the objects of interest from the relational database management system. Additionally, a results container object may be instantiated to store the specified information about the objects of interest.

In another embodiment, the invention provides a computer implemented method of deferring allocation of storage for array elements of objects comprising the steps of: receiving a request to instantiate an object of a class where the class has a definition that specifies a default value for each data member of an array element; allocating storage space for the object without storage space for an array element if the instantiation request does not specify an initial value for any of the data members of the array element; receiving a request to modify a data member of the array element; determining if storage space for the array element has been allocated; if storage space for the array element has not been allocated, allocating storage space for the array element and initializing each data member of the array element to the specified default value; and modifying the data member of the array element as specified in the modification request.

In another embodiment, the invention provides a computer system for storing objects in a relational database comprising: an object-oriented application for receiving a definition of a class of objects that are to be stored in the relational database, the objects being stored in a transient storage; a memory for the transient storage of the objects; an object server for retrieving the objects from the memory and issuing statements in a relational database query language to store data of the objects; and a relational database management system for receiving the statements and storing the data of the objects in persistent storage as relational database tables.

In another embodiment, a method of assigning payment resources to an account balance stored in an RDBMS table includes instantiating a balance adjusting container object that includes an account element identification number, a balance element identification number of a balance element that will be adjusted and an adjusting amount and formulating an RDBMS query to determine whether an entry in an account balance table exists that corresponds to the account element identification number and the balance element identification number. If an entry in the account balance table is found, the found entry in the account balance table is adjusted by the adjusting amount. If no entry in the account balance table is found, then an entry is created in the

account balance table with a default balance and the created entry is incremented by the adjusting amount.

In another embodiment, a method of defining a payment resource for inclusion in a real time transaction processing system includes instantiating a balance element creating container object and including data fields in the balance element creating container object including a balance element identification number field and providing a balance element identification number in the balance element identification number field. A new row is added to a payment resources RDBMS table using the information from the balance element creating container object.

In another embodiment, a method of determining a rate for adjusting a real time balance includes receiving a billing event, the billing event including a rate name, a quantity used, an attribute that describes the event, and an account to be billed. A product table is searched to determine all of the products in the account to be billed and a rate table is searched to determine all of the rates that correspond to the rate name and the products in the account to be billed. Rates are eliminated that do not match the attribute that describes the event and the remaining rates are ordered by descending priority to obtain a rate stack. For each rate in descending order of priority, the maximum portion of the quantity used that can be applied using the rate without violating a credit limit is allocated.

In one embodiment, the invention provides a computer implemented method of performing real time billing of accounts comprising the steps of: during close billing of a billing cycle, receiving a transaction event for an account; locking the account; determining if the account has already been billed in the billing cycle; if the account has not already been billed in the billing cycle, close billing the account without billing the transaction event; posting the transaction event to the account; and unlocking the account. The account may be determined that it has not already been billed by checking if the next bill time of the account is in the past. Typically, the transaction event is a billing event.

In another embodiment, the invention provides a computer implemented real time billing system comprising: a processor; a memory coupled to the processor that stores accounts; a billing process operating on the processor that bills accounts at the end of a billing cycle and upon receiving a transaction event for an account, the billing process locks the account, close bills the account without billing the transaction event if the account has not already been billed in the billing cycle, posts the transaction event to the account, and unlocks the account.

These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures which illustrate by way of example the principles of the invention.

PAGE INTENTIONALLY LEFT BLANK

Brief Description Of The Drawings

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Figure 1 illustrates an example of a computer system that may be utilized to execute the software of an embodiment of the present invention.

Figure 2 shows a system block diagram of the computer system of Figure 1.

Figure 3 shows a block diagram of an embodiment of the invention which provides an object-oriented interface to objects that are stored by a relational database management system.

Figure 4 shows a graphical hierarchical tree that may be utilized to define a class of objects.

Figure 5A shows a structure of a data block utilized to store container objects and Figure 5B shows a container object for storing an object in transient memory.

Figure 6 illustrates relational database tables that may be generated by an object server of the invention to store objects of the class defined in Figure 4.

Figure 7 shows a process of defining and storing an object in a relational database.

Figure 8 shows a graphical hierarchical tree that may be utilized to define a subclass of objects (e.g., subclass of the class shown in Figure 4).

Figure 9 illustrates the additional relational database tables that may be generated to store the additional data members of objects of the subclass defined in Figure 8.

Figure 10 shows a hierarchical tree that may be maintained to store the relationship of classes and their characteristics.

Figure 11 shows a process of generating a subclass.

Figure 12 shows a process of deleting an object.

Figure 13 shows a process of querying the relational database for information about objects of interest through an object-oriented interface.

Figure 14 illustrates a query container object that may be utilized in querying the relational database through an object-oriented interface.

Figure 15 shows a query container object that may be generated by the invention.

Figs. 16A and 16B show a process of generating SQL searches utilizing a query container object.

Figure 17 shows a process of allocating storage space for an array element of an object only if necessary in order to save storage space.

Figure 18A is an illustration of a container object which stores an account object.

Figure 18B illustrates relational database tables that may be generated by the object server to store objects.

Figure 19 is a process flow diagram illustrating a process for adding a new payment source or balance element to an account.

Figure 203 is a process flow diagram illustrating a process for adding a new balance element type to the on line transaction processing system.

Figure 21 is a block diagram illustrating relational database tables used to implement the rating method used in one embodiment of the present invention.

Figure 22 is a process flow diagram illustrating the process that runs when a billing event occurs for an account.

Figure 23 illustrates some relational database tables that may be stored for the purpose of storing real time account balance information.

Figure 24 shows a regular billing process implemented on a computer that sequentially accesses accounts to generate bills for the accounts.

Figure 25 shows a process that may occur when a billing event occurs while the accounts are being billed.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention. An example of a preferred embodiment is illustrated in the accompanying drawings. While the invention will be described in conjunction with that preferred embodiment, it will be understood that it is not intended to limit the invention to one preferred embodiment. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims along with their full scope of equivalents. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

Figure 1 illustrates an example of a computer system that may be used to execute the software of an embodiment of the present invention. Figure 1 shows a computer system 1 which includes a display 3, screen 5, cabinet 7, keyboard 9, and mouse 11. Mouse 11 may have one or more buttons for interacting with a graphical user interface. Cabinet 7 houses a CD-ROM drive 13, system memory and a hard drive (see Figure 2) which may be utilized to store and retrieve software programs incorporating computer code that implements the present invention, data for use with the present invention, and the like. Although the CD-ROM 15 is shown as an exemplary computer readable storage medium, other computer readable storage media including floppy disks, tape, flash memory, system memory, and hard drives may be utilized.

Figure 2 shows a system block diagram of computer system 1 used to execute the software of an embodiment of the present invention. As in Figure 1, computer system 1 includes monitor 3 and keyboard 9, and mouse 11. Computer system 1 further includes subsystems such as a central processor 51, system memory 53, fixed disk 55 (e.g., hard drive), removable disk 57 (e.g., CD-ROM drive), display adapter 59, sound card 61, speakers 63, and network interface 65. Other computer systems suitable for use with the present invention may include additional or fewer subsystems. For example, another computer system could include more than one processor 51 (i.e., a multi-processor system), or a cache memory.

Arrows such as 67 represent the system bus architecture of computer system 1. However, these arrows are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be utilized to connect the central processor to the system memory and the display adapter. Computer system 1 shown in Figure 2 is but an example of a computer system suitable for use with the present invention. Other configurations

of subsystems suitable for use with the present invention will be readily apparent to one of ordinary skill in the art. t to one of ordinary skill in the art.

Figure 3 shows a block diagram of an object-oriented application that stores objects in a relational database. An object-oriented application 101 allows object-oriented creation, manipulation, and searching of objects. In traditional client-server nomenclature, application 101 is the client and it typically operates on a computer system. The computer system may be similar to the one shown in Figure 1.

As objects are created or accessed by application 101, the objects are stored in memory 103. The memory may be generally thought of as transient storage - meaning that the storage is only temporary and is not the permanent storage of the objects. Typically, memory 103 is the dynamic random access memory of the computer system on which the client computer system operates. Of course, memory 103 is not limited to any specific memory-type as it may be cache memory, flash memory, hard drive, floppy disk, and the like.

An object server 105 provides the interface between the object-oriented scheme and the relational database scheme. The object server is a process that translates object-oriented requests into relational database requests (e.g., SQL). Typically the object server operates on the same computer system as the object-oriented application. However, there is no requirement that the object server operate on the same computer system or at the same location (e.g., the two computer systems may be in communication over a network).

The object server sends relational database requests to a relational database management system (RDBMS) 107. The RDBMS stores data in relational tables with columns in the tables representing data of the same type. Although the RDBMS typically operates on a different computer system than the object server, the RDBMS may operate on the same computer system. In traditional client-server nomenclature, RDBMS 107 is the server. In a preferred embodiment, the RDBMS is from Oracle Corporation, Redwood Shores, California.

Object-oriented environments are intended to shield the implementation from the user. More specifically, in database applications an object-oriented environment hides the details of how the objects are stored. This is in stark contrast to traditional relational database applications where it is generally required for a user to know how data is stored in order to formulate queries on the data. It may be beneficial at this point to illustrate an example of how the present invention may store an object in a relational database scheme.

In object-oriented environments, a class defines a group of objects that share the same characteristics. More specifically in this context, each object (or instance) of a class may have the same data members. An object is created by being instantiated as a member of a class.

Figure 4 shows a graphical hierarchical tree that may be utilized to define a class of objects. A graphical hierarchical tree 151 defines a class named "Account" which three data members. In this example, a class for an accounting system will be defined. However, this example is intended to aid the readers understanding of the invention and does not limit the invention to any specific embodiment.

Data member "Name" represents the name of a customer. Data member "Last Billed" represents the date that the customer was last sent a bill. Lastly, data member "Balance" is an array where each element of the array may store three data members. The graphical hierarchical tree indicates that "Balance" is an array by the brackets indicated by arrow 153. The minus sign in box 155 may be activated to collapse the data members of Balance as is commonly done in conventional graphical user interfaces (GUIs).

Each element of the array Balance may include a data member "Name" which represents the currency of the units of the balance, a data member "Current Balance" which represents the current balance, and a data member "Credit Limit" which represents the credit limit. Although this is a very simple example, it illustrates many of the advantages of object-oriented data storage. For example, it is easy for a user to define a class of objects. The user may use pull-down menus to select a type of data member (e.g., integer) . Then the user may drag and drop the new data member on the graphical hierarchical tree at the desired location.

A significant advantage of an object-oriented is inheritance. Inheritance allows one class, the subclass, to inherit or receive all the characteristics of a higher or parent class. Inheritance allows a user to tailor new classes off of an existing class, therefore resulting in a reuse of resources. An example of a subclass of Account will be described in more detail in reference to Figs. 8-10.

A hierarchical tree provides a hierarchy for the data members. Accordingly, the system does not have trouble distinguishing Name which is a data member of each Account object and Name which is a data member of each Balance array element.

In order to provide a thorough understanding of the invention, Figs. 5A and 5B show a container object for storing an object in transient memory. However, it should be understood that the actual implementation of the container object is not apparent to the user. Figure 5A shows the structure of a data block that may be utilized to store a container object.

A data block 201 includes four fields for storing the following information:

Name - indicates the name of the data in the data block

Type - indicates the data type of the data in the data block

Value - stores the value of the data in the data block

Element Id - indicates the id of an element of an array

In order to better understand the data blocks, one should refer to Figure 5B which shows a container object 203 which stores an object of the class defined in Figure 4. The container object includes a main container 205 and two subcontainers 207 and 209.

Main container 205 includes a header 211 which is for storing any information about the main container (e.g., number of subsequent data blocks and their location in memory). Following the header, there are one or more data blocks that have the same structure as the data block shown in Figure 5A.

Referring to a data block 213, the Name of the data block is <Sys Id> indicating that this data block stores a system id for the object, which may be thought of as the name of the object from the system's point of view. In a preferred embodiment, the Sys Id includes a type string, id number, database number, and revision number.

In data block 213, the Type is "Sys Id" indicating that this data block identifies an object. The Value of the data block stores 3456 which is a number which will be utilized to join relational tables storing data members for this object. Accordingly, this number will be referred to as the "Id" when discussing the relational database tables storing the object. As mentioned in the preceding paragraph, the Sys Id may be a compound data type including an id number. Thus, in some embodiments, the Id (e.g., 3456) is a portion of the Sys Id. Lastly, the Element Id is "N/A" (not applicable) as this data block does not store an element of an array. Thus, it should not matter what is stored in the Element Id field.

Although data block 213 contains the same fields as the other blocks, it is a unique block as it identifies an object. Accordingly, a better understanding of the data blocks will be achieved from a detailed discussion of the subsequent data blocks.

A data block 215 stores a data member of the object. The Name field contains "Name" which is the name of the first data member of the class that is defined in Figure 4. The Type of the data block is "String" which indicates the Value field stores a string which is shown as "John Doe." As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 217 stores another data member of the object. The Name field contains "Last Billed" which is the name of the second data member of the class (see Figure 4). The Type

of the data block is "Date" which indicates the Value field stores a date which is shown as a string for April 1, 1997. As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 219 stores an element of the array "Balance" utilizing subcontainer 207. The Name field contains "Balance" which is the third data member (an array) of the class (see Figure 4). The Type of the data block is "Array" which indicates the Value field stores an element of an array. The Value field stores an element of the array by storing a pointer to subcontainer 207 which stores the data members of the element. The Element Id field contains the number 840, which according to the International Standards Organization (ISO) specifies U.S. dollars. Although in this particular example the Element Id field has a specific meaning in addition to identifying an array element, typically the Element Id field acts to identify the array element. The Element Id field will be utilized to join relational tables storing data members for this object.

Subcontainer 207 contains three data blocks which indicate that this object has a Current Balance of 500 U.S. Dollars and a Credit Limit of 10,000 U.S. Dollars. The first data block indicates the currency of the other two data blocks.

A data block 221 stores another element of the array "Balance" utilizing subcontainer 209. Thus, each element of the array may include a data block and a subcontainer. Substructures are implemented similar to arrays and may be thought of as an array with one element (the Element Id may be inapplicable though). Of course, there is no limit to the complexity of the objects stored under this implementation. For example, an array may contain an array, which contains an array, and so on.

Although the data blocks were shown in the same order as the data members were represented in the graphical hierarchical tree defining the class shown in Figure 4, it is not necessary that the data blocks be stored in any particular order. For example, when an element is added to an array, the data block for the new element may be added to the end of the main container.

Putting the container object shown in Figure 5B into perspective, the container object is typically stored in memory 103 shown in Figure 3. Therefore, the container objects are the method of communicating data (along with API calls) between the object-oriented application and the object server. Now it may be beneficial to describe the way the object will be stored in the relational database as relational tables.

Figure 6 illustrates relational database tables that may be generated by the object server to store objects. A relational database table (Account_T) 251 is the main relational table for the

objects. As shown, the table includes columns entitled "Id," "Name" and "Last_Billed." The Id refers to a number that will be utilized to identify this object in the relational database and therefore, join relational database tables. In some embodiments, a Sys_Id column may be utilized in place of the Id column where the value in the Sys_Id column is compound data type to identify the object where Id is included in the Sys_Id. The Name and Last_Billed columns store data members of the object (see Figure 4).

A relational database table (Account_Balance_T) 253 stores elements of the array "Balance." As shown, the table includes a column entitled "Id" which stores the id of the object for which the data in this table belongs. As one familiar with relational databases will recognize, the Id will be utilized to join relational database tables 251 and 253. Table 253 also includes a column entitled "Element_Id" which designates the element id of this element. In the instant case, the element id designates the currency of the data in the balance element. Lastly, table 253 includes columns entitled "Name," "Current_Balance" and "Credit_Limit" which may be data members of an element of the array Balance (see Figure 4). In some embodiments, the Name column is located in a different relational database table but it is shown here in table 253 for simplicity.

Additionally, the value in the Element_Id column is utilized to identifyBalance array elements. Each Balance array element is stored in the relational database as a row in a relational table. Accordingly, storage space need only be allocated for those array elements that have been declared. In some embodiments, the allocation of storage space for array elements is deferred until needed in order to save storage space. This process will be described in more detail in reference to Figure 17.

It should be readily apparent that by storing data members of objects in relational database tables, one may use conventional relational database management systems to query the relational database tables. Other features of the invention that will be described in more detail below are that array elements are easily added to an object by the addition of another row of a relational database table, subclasses may be defined which inherit the data members of a higher or parent class, and array elements of objects need not be allocated until actually utilized (this will also be called "lazy allocation of array elements").

Figure 7 shows a process of defining and storing an object in the relational database. At step 301, the user defines a class for the object. The class may be parent class or a subclass that inherits characteristics from another class. For the moment, assume the user defines the class shown Figure 4.

The object-oriented application analyzes the class definition and generates a hierarchical tree for parsing at step 303. The hierarchical tree includes information for mapping an object's

data members to columns in a relational database table. Accordingly, the hierarchical tree is parsed in order to map from the object-oriented scheme to the relational database scheme. The hierarchical tree also includes information about inheritance between or among classes as will be shown in Figure 10.

At step 305, the object-oriented application instructs the object server to generate the requisite relational database tables. The object server then sends SQL calls to the RDBMS to generate the specified tables. At this point, the relational database tables have been generated but they do not contain any data.

In order to store an object in the relational database, the object-oriented application instantiates a container object at step 307. The container object will store the data members of the object while it is in transient memory. The user sets the value of data members of the object at step 309.

When a class is being defined, the user may specify certain characteristics of the data members. These characteristics are stored in the hierarchical tree that is utilized to map between the object-oriented scheme and the relational database scheme. For example, the user may specify that a data member is mandatory for an object of the class. The system then verifies that every object created of that class contains a value for that data member. Conversely, the user may specify a data member is optional so that the data member is not required for every object of the class.

The user may also specify that data members have default values. These default values will also be stored in the hierarchical tree and may be utilized to initialize an optional data member if no initial value is specified. If no value is specified for a mandatory data member, the user is informed that this is an error.

Additionally, default values may be utilized for the lazy or late allocation of array elements. An array element with a default value for each data member of the array element need not be allocated storage space in the container object (and therefore the relational database tables) until the array element is accessed or modified. Thus, only when an array element is actually needed is the storage space allocated. This feature of the invention provides significant storage savings.

At step 311, the object-oriented application instructs the object server to store the object in the relational database. The application sends an API call to store an object along with a reference or pointer to the container object. The data in the container object includes the values for the data members of the object.

The object server references the container object and generates SQL calls to the RDBMS to store the data in the appropriate relational database tables. The RDBMS will generate the necessary new rows and store the data in the appropriate columns of the relational database tables.

The above has described a simple example where the defined class is a highest level class (i.e., is not a subclass so it does not inherit characteristics from a parent class). One of the powerful features of the invention is that subclasses may be defined and objects of those classes inherit characteristics (e.g., data members) from a parent class and may be stored in the relational database.

Figure 8 shows a graphical hierarchical tree that may be utilized to define a subclass of objects. A graphical hierarchical tree 351 is shown that is divided into two halves by a dashed line 353. The top half of the graphical hierarchical tree is the same as shown in Figure 4. Thus, the subclass being defined inherits the characteristics of the class "Account."

As shown, the subclass adds a new data member "Address" which is a substructure including a data member named "Street," a data member named "City" and a data member named "Phone" which is an array. Each element of the array Phone may have a data member named "Type" and a data member named "Number." As an example, the Type data member may indicate the Number of the Phone element is a home phone number.

The container object for storing an object of the subclass may look similar to the one shown in Figure 5B with an additional data block in the main container for the Address that points to a subcontainer that stores the data members for the Address. This subcontainer will also have a data block that points to a subcontainer for each element of the array Phone.

Figure 9 illustrates the additional relational database tables that may be generated to store the additional data members of objects of the subclass. A relational database table (Account_Address_T) 401 stores data for the two nonarray data members of Address. As shown, a single address has been stored in the relational table. The Id column indicates that this is an address for John Doe as it has the same id number. The relational database system will utilize this column to perform joins on the relational database tables.

A relational database table (Account_Address_Phone_T) 403 stores data for each element of the array Phone. As shown, there are two elements present with element being a row in the relational database table. Each row includes an Id and an Element_Id, in addition to the data member columns. Since each element of an array is represented as a row in a relational table, it is easy to add new elements to the array as new rows since RDBMS are designed to add new data in this fashion.

Although adding new rows to relational database tables is quite easy, this is not the case with adding new columns. For this reason, in some embodiments, the data members defined in subclasses are constrained to be stored as substructures and arrays by the RDBMS. As mentioned earlier, substructures may be thought of conceptually as a one element array. Both substructures and arrays are represented in the relational database as new tables.

Figure 10 shows a simple representation of the hierarchical tree that may be maintained to store the relationship of the classes and their characteristics. A hierarchical tree 451 includes a node for each related class and subclass. The topmost node A represents the parent class. The nodes X and Y are subclasses of the parent class and therefore inherit characteristics of the parent class. Node Z represent a subclass of the class X so it inherits characteristics of both the parent and grandparent classes.

At each node is stored information to map each data member of the class to one or more columns in the relational database tables. For example, a mapping from the data member Current Balance of the subclass to the relational database column Account_Balance_T.Current_Balance (see Figure 6). Additionally, each node will store other characteristics of the data members of objects of the class (e.g., default values).

The hierarchical tree allows a user to define a class hierarchy that is arbitrarily complex and each class itself may be arbitrarily complex. For example, at any node in the hierarchical tree, there may be any number of data members, structures, arrays or nesting of these data types (limited only by the capacity of the computer system or software).

Although Figure 7 shows a process for both defining a class and storing an object of the class, these operations may be performed separately as will be demonstrated by a discussion of Figure 11 which shows a process of generating a subclass. At step 501, the user defines a subclass as was discussed in reference to Figure 8. The object-oriented application generates a hierarchical tree for parsing. This hierarchical tree includes characteristics of the subclass and the parent class from which it inherits.

At step 505, the object-oriented application determines a difference between the hierarchical tree for this subclass and the hierarchical tree for its immediate parent class. Conceptually, this would identify the new data members as is shown on the bottom half of Figure 8.

The object-oriented application issues an API call to the object server generate the new tables for the differences between the hierarchical tree for the parent class and subclass at step 507. The object server issues the appropriate SQL calls to generate the additional relational database tables for objects of the subclass. Data members for the objects which are inherited

from the parent class(es) will be stored in the relational database tables that were generated for each class. Thus, data members that are common to related objects will be stored in the same relational database tables. This provides a number of advantages including the following.

An advantage of the invention is that one is not limited to any one class during queries. For example, one may search for all objects that were billed last on 4/1/1997. The RDBMS and the object server may return objects that, although related, are not the same class. Some objects may be of a parent class while other objects may be of a subclass, or a subclass of a subclass. The object server returns a list of objects in a container object that satisfy the query.

There are numerous API calls that may be issued to the object server. A detailed description of all the API calls is not necessary to understand the invention. However, it may be beneficial to illustrate a process of utilizing one of the available API calls. The API call that will be discussed deletes an object from the relational database.

Figure 12 shows a process of deleting an object. At step 551, the object-oriented application instantiates a container object in transient memory. It is presumed that it is already known which object should be deleted (e.g., results from a search). The object to be deleted will be identified by the Id (or Sys Id), which is the id given to the object by the system.

At step 553, the object-oriented application inserts the Id of the object to be deleted into the container object. The application then issues an API call to the object server at step 555 instructing it to delete the object identified by the container object.

The object server then verifies that the Id in the container object specifies a valid object at step 557. After verification, the object server generates the appropriate SQL calls to delete the data in the relational database at step 559. In a preferred embodiment, the multiple SQL calls that may be necessary to delete an object from the relational database tables are performed as one atomic operation. In other words, the multiple SQL calls are performed as one operation conceptually so the user is not able to access a partially deleted object in the relational database. The one or more SQL calls to delete the oject are received by the RDBMS and executed to perform the deletion .

The invention utilizes the power of relational database management systems to access objects that are stored in the relational tables. Figure 13 shows a process of querying the relational database for information about objects of interest through an object-oriented interface. Before describing the figure, it may be beneficial to review some fundamentals of relational databases.

One of the most popular relational database query languages is SQL. The basic format of an SQL query is the following:

SELECT {columns} FROM {tables} WHERE {conditions}

As an example, suppose using the sample database described in Figs. 4-6 that one wants to know the names of people who have accounts that were last billed on April 1, 1997 and have a current balance greater than $100. An SQL query for this information may resemble the following:

SELECT Acount_T.Name FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100 and Account_T.Id = Account_Balance_T.Id

As should be apparent, the user is required to know the way the data is stored in the relational database to form a correct query. For example, the last condition that specifies that the ids of the two tables are the same specifies how to link the two tables and is called a "join" operation.

Referring again to Figure 13, the object-oriented application receives an object-oriented query at step 601. The object-oriented query will typically specify values of data members of objects of interest. The application instantiates a container object at step 603. As with any flowcharts depicted herein, there is no implied order of the steps simply by the way they are presented.

In general, the application exchanges data with the object server through container objects stored in transient memory. Since this container object is for performing a query, we will call it a query container object. Figure 14 illustrates a query container object that may be utilized in querying the relational database. As shown, a query container object 651 includes a query template, args (or arguments) array and a results array. The query template contains a query with gaps that are filled in by the args array and results array. The query container object will be described in more detail in reference to Figure 15.

At step 607, the object-oriented application instructs the object server to perform the query. The application sends an API call to the object server that references the query container object. The object server analyzes the information in the search object container and generates SQL calls (or queries) for the RDBMS at step 609. This process will be described in more detail in reference to Figs. 16A and 16B. After step 609, the object-oriented query has been translated into a relational database query.

In order to perform the query, the object server may send multiple SQL queries to the RDBMS. Once the object server receives the desired results, the object server instantiates a new container object and stores the results of the query in the container object. Although the results may be placed in the search object container, instantiating a new container object allows the original search object container to be unmodified which may be preferable.

At step 613, the object-oriented application retrieves the results of the query from the container object that stores the results in transient memory and displays the results. The results may be displayed any number of ways but are preferably displayed in an object-oriented fashion.

When a user specifies a query in an object-oriented system, the user specifies the desired data members of the objects of interest (or the whole object) along with the conditions that determine the objects of interest. Thus, the relational database query described above (i.e., SELECT Acount_T.Name FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100 and Account_T.Id = Account_Balance_T.Id) in an object-oriented environment would conceptually be "retrieve the Name data member of all objects of class Account where the Last Billed is equal to April 1, 1997 and the Current Balance is greater than 100 dollars." The form of the object-oriented query may vary and the form of some embodiments will be described in the following paragraphs. However, it is important to notice that with an object-oriented query, the user is not required to know exactly how the objects are stored in memory. This is in stark contrast to conventional relational database systems as is illustrated by the SQL query.

Once the user has entered an object-oriented query, the object-oriented application fills in the query container object. Thus, the application inserts the query template, args array and results array into the query container object. The query template is an SQL-like query that includes object-oriented information as follows:

SELECT { } FROM {class} WHERE {conditions without joins}

The query container object includes a results array which is an array which holds the data members and/or objects that should be returned from the relational database. Since the results array includes what is typically included in the SELECT clause, this clause may just have a place holder like 'X' as in some embodiments.

The FROM clause specifies the class of objects that should be searched. If a parent class is specified, then subclasses will satisfy the query also. Lastly, the WHERE clause specifies conditions like Data_Member1 = Value1. Data_Member1 and Value1 are determined from the first element in the args array. Thus, the first element in the args array would specify an element

of the Name "Last Billed," Type "Date" and a Value of "4/1/1997." The args array fills in the missing values of the query template which may be as follows:

SELECT X FROM Person WHERE Data_Member1 = Value1 and Data_Member2 > Value2

The SELECT clause has an X as a place holder so the query is easier to parse and looks better, but the results array will specify the data members and/or objects requested. As indicated above, the WHERE clause will be filled in from the args array but it should be noted that there are no joins specified.

Figure 15 shows a query container object that may be generated by the invention for the above search. A query container object 701 includes a main container 703 and subcontainers 705, 707, 709, and 711. A data block 713 in the main container stores the query template as discussed above. Data blocks 715 and 717 store the pointers to the elements of the args array. Data block 715 points to subcontainer 705 which includes a data block 719. Data block 719 includes the information to fill in one of the conditions of the query template (the operator '=' is already in the query template). The Name field indicates that Data_Member1 is the "Last Billed" data member and the Value field indicates that Value1 is "4/1/1997" (see Figure 5A for details on the data blocks).

Similarly, data block 717 points to subcontainer 707 which includes a data block 721. Data block 721 points to an array element for the Balance array. Data block 721 points to a data block 723 which includes the information to fill in another of the conditions of the query template (the operator '>' is already in the query template). The Name field indicates that Data_Member2 is the "Current Balance" data member and the Value field indicates that Value2 is 100.

A data block 725 stores an element of the results array. Data block 725 points to subcontainer 711 which includes a data block 727 that indicates the results the query requests. As you may recall, the user wanted the name of the individual. Data block 727 indicates this by having a Name field that specifies the "Name" data member and a Value field that is blank indicating this information should be the results obtained by the query.

Figs. 16A and 16B show a process of generating SQL searches utilizing a query container object (see also step 609 in Figure 13). In some embodiments of the invention, the process shown is performed by the object server. As will be described, what may appear to be a single query may be implemented in multiple SQL queries. However, the invention may ensure that the minimum joins that are necessary are performed as joins are a relatively expensive relational database operation.

The object server will be constructing SQL queries for the RDBMS that retrieve the data of interest. In order to aid the reader's understanding, the process will be described as implementing the query that has been described above where one is searching for the Name data member of objects of class Account that were Last Billed on April 1, 1997 and have a Current Balance greater than 100 dollars.

At step 751, the column for the object Id is put in the SELECT clause. The object server utilizes the hierarchical tree to map the object Id to the associated column in the relational database. The following shows the SQL query that has thus far been generated:

SELECT Account_T.Id FROM WHERE

The object server identifies relational database tables that will be needed to satisfy this query and inserts the tables in the FROM clause at step 753. The object server first identifies all the relational database tables that contain columns to which the data members in the args array are mapped. This involves an analysis of the hierarchical tree and the args array in the query container object. Additionally, the object server determines if any additional tables that are needed for any joins that may be necessary to link the columns in different tables. The following shows the SQL query with the requisite tables:

SELECT Account_T.Id FROM Account_T, Account_Balance_T WHERE

At step 755, the object server inserts conditions from the query specified by the query container object in the WHERE clause. The object server identifies the data members in the args array and uses the hierarchical tree to map the data members to the appropriate columns in the relational database tables. The comparators (e.g., '=' and '>' in this example) are retrieved from the query template. The values to complete the conditions are retrieved from the args array. The following shows the SQL query with the conditions for this example:

SELECT Account_T.Id FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100

Since SQL specifies that joins are explicitly placed in the query, the object server then uses the hierarchical tree to identify the requisite joins and insert the joins in the WHERE clause at step 757. The process of identifying the requisite joins is similar to the process of identifying all the relational tables for the query at step 753. Accordingly, the steps may be performed at the same time.

The object server identifies the joins that are required by parsing the hierarchical tree to determine the relational database tables (e.g., column) to which each of the data members specified in the query template and args array. For each of the different tables, the object server

has to generate a database join to link the two tables (unless that join has already been identified). The following shows the SQL query with the necessary join:

SELECT Account_T.Id FROM Account_T, Account_Balance_T WHERE Account_T.Last_Billed = "4/1/1997" and Account_Balance_T.Current_Balance > 100 and Account_T.Id = Account_Balance_T.Id

Although in this simple example it is not required, the object server may need to generate multiple joins to link two tables and the joins may include tables that are not immediately apparent from an inspection of the query template and the args array. For example, while the object server is parsing the hierarchical tree, it may identify a table "between" two previously identified tables, therefore creating joins to that intermediate table. By analyzing the hierarchical tree, the invention is able to generate an SQL query that utilizes the minimum number of tables and joins.

Once the SQL query is generated, the object server issues the query to the RDBMS at step 759. The SQL query produced requests the object Ids of all objects of the class Account that satisfy the query. The desired Name data members will identified in a separate SQL query described in Figure 16B.

Referring now to Figure 16B, the object server receives from the RDBMS the object Ids of the objects that satisfy the SQL query at step 761. The objects that satisfy the query may include objects that are of subclasses of the Account class. Thus, the invention fully supports inheritance during searching. The user could also have specified a subclass and the SQL query would have joined in a table that would select only objects of that subclass (and those that inherit from that subclass).

At this point, the object server now knows which objects satisfy the query but the query container object has a results array that specifies the Name data member is desired. Therefore, the object server generates a second SQL query to retrieve the desired data members.

At step 763, the object server inserts the desired columns identified in the results array in the SELECT clause. In this example, the results array specifies the Name data member of the class Account. The object server uses the hierarchical tree to map the data member to the appropriate column or columns in the relational database and produces the following initial SQL query:

SELECT Account_T.Name FROM WHERE

The object server identifies the tables for this query and inserts them in the FROM clause at step 765. With this simple example, the only relational database table that is required is Account_T so the SQL query becomes the following:

SELECT Account_T.Name FROM Account_T WHERE

At step 767, the object server inserts the desired object Ids into the WHERE clause. Although many objects (or none) may satisfy a query, assume that the only one that does has an Id data member of 3456. The SQL query will then become the following:

SELECT Account_T.Name FROM Account_T WHERE Account_T.Id = 3456

Since the first SQL query found objects that satisfied the query, the same conditions do not have to be put in the second SQL query. At step 769, the object server uses the hierarchical tree to identify any joins that may be required to be inserted in the WHERE clause. With this example, no joins are necessary so the SQL query remains unchanged. The object server then issues the SQL query to the RDBMS at step 771.

As mentioned earlier, one of the advantages of the invention is that a user is able to specify characteristics of data members including default values. Default values may be used to defer allocation of storage space (e.g., memory) for array elements ("lazy allocation"). The default values for array elements are stored in the hierarchical tree that is utilized for parsing and the following discusses how allocation of storage space is deferred.

Figure 17 shows a process of allocating storage space for an array element of an object only if necessary in order to save storage space. If the class (or subclass) definition defines a default value for each data member of an array element and an object of that class is instantiated without specifying a value for a data member of an array element, the system does not allocate storage for the array element. Subsequently, when the system receives an instruction to access or modify a data member of the array element at 801, the system checks if the array element has been allocated as shown at step 803.

If the array element has not yet been allocated storage space, the system allocates storage space for the array element and assigns each data member of the array the appropriate default value from the hierarchical tree at step 805. Once the system has verified that storage space for the array element has been allocated, the system may access or modify the data member (e.g., increment) as instructed at step 807.

As was described in Figure 5A, information is transferred by a user to and from the object server using container objects. Each object within a container object includes a number of data blocks. A data block includes four fields for storing the following information:

Name - indicates the name of the data in the data block

Type - indicates the data type of the data in the data block

Value - stores the value of the data in the data block

Element Id - indicates the id of an element of an array

Figure 18A is an illustration of a container object 1803 which stores an account object. The account balances for each payment source are represented as a balance array, as is further described below. The container object includes a main container 1805. Main container 1805 includes a header 1811 which stores information about the main container (e.g., number of subsequent data blocks and memory location). Following the header, there are one or more data blocks that provide account information.

Referring to a data block 1813, the Name of the data block is <Sys Id> indicating that this data block stores a system id for the object, which may be thought of as the name of the object from the system's point of view. In a preferred embodiment, the Sys Id includes a type string, id number, database number, and revision number.

In data block 1813, the Type is "Sys Id" indicating that this data block identifies an object. The Value of the data block stores 3456 which is a number which will be utilized to join relational tables storing data members for this object. Accordingly, this number will be referred to as the "Id" when discussing the relational database tables storing the object. Lastly, the Element Id is "N/A" (not applicable) as this data block does not store an element of an array. Thus, it should not matter what is stored in the Element Id field.

A data block 1815 stores a data member of the object. The Name field contains "Name" which is the name of the first data member of the class. The Type of the data block is "String" which indicates the Value field stores a string which is shown as "John Doe." As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 1817 stores another data member of the object. The Name field contains "Last Billed" which is the name of the second data member of the class. The Type of the data block is "Date" which indicates the Value field stores a date which is shown as a string for April 1, 1997. As the data block does not store an element of an array, the Element Id field is not applicable.

A data block 1819 stores an element of the array "Balance" utilizing subcontainer 1807. The Name field contains "Balance" which is an array because the type field of the data block indicates that it is an array. The balance array is used to keep track of each different type of payment source for the account. Instead of containing an actual value or set of values as would normally be the case for an array, the value field contains a memory pointer to a subcontainer 1807 which stores the data for the element of the array balance that is represented by data block

1819. The Element Id field contains the number 840. In this example, 840 functions in two ways. First, 840 is the array index. Thus the array element balance (840) is distinguished from and array element balance (920). 840 also happens, according to the International Standards Organization (ISO), to specify U.S. dollars. The Element Id field is used to join relational tables storing data members for this object.

This special relationship of arrays to memory is significantly different from the way arrays are generally stored in memory. Generally, all of the elements of an array are assigned locations in memory and the index of the array points to the specific memory location that is assigned to the element of the array that corresponds to the index. If certain elements of the array contain no data, then those memory locations are reserved, but blank. This can result in a very large amount of memory being allocated or reserved in order to add a single element to an array.

This can be clearly seen in the present example, where every account has a balance array. In a typical on line transaction processing system, there may be millions of accounts. Additionally, each balance array as shown is actually an array of structures. For every balance element, there exists a current balance, and a credit limit. Without the structure shown in Figure 18A, adding a new payment source, and therefore a new balance element, would be extremely expensive from a memory standpoint. Each balance structure (of which there is one for every account) would have to have the additional fields added to it, at great cost in memory.

In contrast, the scheme shown in Figure 18A provides a flexible or "lazy" allocation of memory to elements in the array. For each account, only those array elements that represent payment resources included within that account are ever assigned memory. Each balance array element is included in the account container as a data block and the value of the data block points to a memory location of a container that includes the name, current balance, and credit limit for the balance array element. The amount of allocated memory that is saved is large, especially if a large number of accounts exist and certain balance elements are shared by a relatively small number of account holders, and/or if a large number of payment resources exist..

Subcontainer 1807 contains two data blocks which indicate that balance array element 840 for the account in container 1805 has a Current Balance of 500 U.S. Dollars and a Credit Limit of 10,000 U.S. Dollars. The name of the currency of the two data blocks is indicated by the array element index, which is also called the balance element ID. Other nonstandard indices may be used to track resources such as frequent flier miles. A data block 1821 stores another element of the array "Balance" utilizing subcontainer 1809.

Putting the container object shown in Figure 18A into perspective, the container object is typically stored in memory 103 shown in Figure 20. Therefore, the container objects are the method of communicating data (along with API calls) between the object-oriented application and

the object server. As is described in detail in U.S Patent Application Attorney Docket No. PORTP001 filed May 14, 1997, an hierarchical list is used to map the data included in data objects placed in containers to the various fields of a relational database. Now it may be beneficial to describe the way that payment resources are stored in the relational database as related tables.

Figure 18B illustrates relational database tables that may be generated by the object server to store objects. A relational database table (Account_T) 1851 is the main relational table for the objects. As shown, the table includes columns entitled "Sys_Id," "Id," "Name" and "Last_Billed." The Sys_Id refers to an identification of the object given by the system. The Id refers to a number that will be utilized to identify this object in the relational database and therefore, join relational database tables. The Name and Last_Billed columns store data members of the object.

A relational database table (Account_Balance_T) 1853 stores elements of the array "Balance." As shown, the table includes a column entitled "Id" which stores the id of the account object for which the data in this table belongs. As one familiar with relational databases will recognize, the Id will be utilized to join relational database tables 1851 and 1853. The Id may also be utilized to join relational database table 1853 with other relational database tables that are included in the database, such as an account system. Table 1853 also includes a column entitled "Element_Id" which designates the element id of this element. In the instant case, the element Id indicates which element of the balance element array is stored in the row and also designates the currency of the data in the balance element. Lastly, table 1853 includes columns entitled "Current_Balance" and "Credit_Limit" which are the data members of each element of the array Balance in this example.

It should be readily apparent that by storing data members of objects in relational database tables, it is possible to use conventional relational database management systems to query the relational database tables. Thus, the benefits of relational database system searching are enjoyed while the object oriented interface simplifies adding payment resources and adjusting payment source balances.

A relational database table (Resources_T) 1854 keeps track of the various payment resources that have been defined for the system. As shown, the table includes a column for each balance element ID. The resource name for each balance element is also included in a separate column, as is a graphical user interface (GUI) name. The GUI name is used in systems where special GUI information is provided to the user in certain circumstances. Another column defines the rounding characteristics for each balance element.

Adding a new type of payment source to the system involves adding a new array element to the balance element array. This is accomplished in memory by adding a row to table 1854. Adding an existing payment source to an account object involves adding a new row to table 1853. The processes for adding a new type of payment source and for adding an existing payment source to an account object are described in further detain in Figures 19 and 23. An important feature of the memory system described above is that memory is not allocated to arrays such as the balance element array until it is needed. This is the so-called "lazy allocation" of memory.

Figure 19 is a process flow diagram illustrating a process for adding a new payment source or balance element to an account. The process starts at 1900. In a step 1902, a balance adjusting container object is instantiated that contains an account ID number, and one or more balance element ID's of the balance element that is to be adjusted, and an adjustment amount for one or more balances. (Note that a single balance element, such as dollars may have several balances associated with it such as a credit balance and a credit limit.) Next, in a step 1904, the container object is sent to the object server. In a step 1906, the object server formulates a query to find the one or more entries in Account_Balance_T that corresponds to the account ID number and balance element specified in the balance adjusting container object. If an entry is not found, then the object server formulates a query for the table Resources_T to see if the balance element exists in Resources_T. If the balance element does not exist in Resources_T, then it is undefined system wide and so an error message is sent in a step 1907 and then the process ends at 1920. If the balance element exists in Resources_T, then control is transferred to a step 1908 and a row is added in Account_Balances_T with default values in the columns and control is transferred to a step 1910 where the newly created entry balances are incremented by the amounts specified in the balance adjusting container object. If an entry is found in Account_Balance_T in step 1906, then control is transferred directly to step 1910 and the found entry balances are incremented by the amount specified in the balance adjusting container object, which may be either a positive or a negative number Once step 1910 is completed, the process ends at 1920.

The process described in Figure 19 is considerably simplified from the standpoint of the application in that it works whether or not a particular balance exists for an account. If the balance does not exist, it is simply added with a default value of zero and then is incremented in step 1910.

Figure 20 is a process flow diagram illustrating a process for adding a new balance element type to the on line transaction processing system. The process starts at 2000. In a step 2002, a balance element type creating container is instantiated. In a step 2004, A balance element ID, a resource name, a GUI name, and a rounding amount are added to the balance element type creating container as data objects. In a step 2005, the object server checks to make sure that all

of the fields required to add a new balance element type are in the container. If not, then an error message is generated. In a step 2006, the object server then adds a new row to the Resources_T table using the information from the balance element type creating container object. The process ends at 2008.

Thus, adding a new type of payment source to the system is done by instantiating a balance element type creating container object and providing the necessary information in data blocks contained in the balance element type creating container object. Adding a new type of payment source to an account is accomplished by instantiating a balance adjusting container object. and providing the necessary information in the data blocks contained in the balance adjusting container object.

So far, a system and method for defining payment resources and assigning payment resources (which will now be referred to as "resources") to accounts has been described. Now, a system and method for increasing and decreasing the balances of different payment resources defined for an account according to a flexible rating method will be discussed.

The following definitions apply to the rating method described below:

Product - A product is an asset that is bought by a customer that may influence or determine amount of resources that the customer will be charged for consuming certain rated quantities. An example of a product would be a certain calling plan that entitles the user to pay 1.00 per hour for a certain rated quantity. Another product could be a plan that entitles the user to pay 2.00 per hour for the same rated quantity.

Rate - A rate is something that determines the cost of a quantity, that is a rate determines how a quantity will impact customer balances when it is consumed. For example, a rate used by an internet service provider may be "type A connect time". The rate determines the cost of a quantity of hours or other consumable according to its impact on various balances.

Figure 21 is a block diagram illustrating relational database tables used to implement the rating method used in one embodiment of the present invention. A Product_t table 2102 contains a list of each product that is available. Product_t table 2102 includes Product ID column that identifies the product, a Name column that provides the name of the product, and a number of attribute columns that may describe things about the product such as when the product may be purchased or the type of customer who is eligible to purchase the product.

A Rate_t table 2104 contains a list of all of the rates that may be charged for a quantity. A Rate_t table 2104 includes a Rate ID column that identifies the rate, a product ID column that identifies the product associated with the rate, and a rate name column that gives the name of the

rate. It should be noted that rate names are generally not unique. the same rate name may be repeated for many rate ID's. Rate_t table 2104 also includes a priority column that indicates the priority of each rate, a default flag column that indicates whether each rate is eligible as a default rate, and an attribute column that indicates such attributes of the rate such the time of day when it is applicable.

An ACCT_t table 2106 contains a list of the accounts. ACCT_t table 2106 includes an Account ID column that identifies the account and various account attributes such as the customer name and when the customer was last billed.

An ACCT_BAL_t table 2108 contains a list of all account balance elements that are contained in the various accounts. ACCT_BAL_t table 2108 includes an Account ID column that identifies the account, a Balance Element ID column that identifies the balance element type, a current balance column and a credit limit column.

An ACCT_PROD_t table 2110 contains a list of all of the products that are contained in the various accounts. An ACCT_PROD_t table 2110 contains an Account ID column that identifies the account, An element ID column that provides an index for the ACCT_PROD_t table, and a Product ID column that identifies the product that is included in the account identified by the Account ID.

A RATE_BAL_IMPACT_t table 2112 contains a list that describes the way that each rate will impact the various balances that are in an account. RATE_BAL_IMPACT_t table 2112 includes a rate ID column that identifies the rate, an Element ID column that provides an index for the RATE_BAL_IMPACT_t table, a Balance Element ID column that determines the balance element that is impacted, a fixed operand column that determines a minimum amount of the balance that is charged, a scaled operand column that determines the rate that the balance is charged after a free quantity, a free quantity column that determines the quantity that is provided for the fixed operand amount, and a type column that is used to describe the type of charge that is made (e.g. a debit, credit, or rebate.)

In one embodiment, the fixed operand, scaled operand, and free quantity are used according to the following cost formula:

cost = fixed operand + scaled operand * (quantity - free quantity).

It should be noted that in other embodiments, other formulas are used and other columns are provided in the RATE_BAL_IMPACT_t table to provide the operands required. It should also be generally noted that the columns in the tables shown are not intended to be all of the columns in each table, and shown for the purpose of illustrating the present example.

Figure 22 is a process flow diagram illustrating the process that runs when a billing event occurs for an account. The process starts at 2200. In a step 2202, a billing event is received by the system. The event includes a rate name, a quantity used, attributes that describe the event, and the account that is to be billed. Next, in a step 2204, the ACCOUNT_PROD_t is searched to determine all of the products that are owned by the account. In a step 2206, the RATE_t table is searched to retrieve all of the rates that match the rate name and the found products for the account. Thus, only rates that have a rate name that corresponds to the rate name of the event and that are associated with a product that is owned in the account are retrieved.

In a step 2208, all rates that do not have attributes that match the attributes of the billing event are discarded. What is left is a list of usable rates. The usable rates are then ordered by descending priority to obtain a rate stack in a step 2210. Next, in a step 2212, the quantity billed is allocated to each rate in order of priority and the applicable resources are adjusted according to the cost that is derived from the RATE_BAL_IMPACT_t table as described above. Each rate is allocated the maximum quantity that can be applies without exceeding a credit limit. If all of the rates are exhausted before the entire quantity billed is allocated, then control is transferred to a step 2214 and highest priority rate in the stack that has a default flag that indicates that the rate is an eligible default rate is used to allocate the remainder of the quantity. The process then ends at 2216. If the entire quantity billed is allocated in step 2214, the process ends at 2216 directly.

The above described process, combined with the tables shown in Figure 21 provides an extremely adaptable rating and billing system. In particular, the RATE_BAL_IMPACT_t table enables a single rate to impact multiple resources or even the same resource independently. For example, if the billing event is one hour of Type A connect time, then the customer can be charged hours, dollars, or any other resource, and can also be given free time, frequent flier miles or any other resource. In fact the customer could be billed dollars and rebated dollars. The cost formula shown above also allows the user to be billed a flat fee, an hourly rate, or some combination of a flat fee. Different rate brackets (similar to income tax brackets) applicable to different quantities can also be defined. Rates are prioritized and charged according to available credit limits and a default rate is applied when to available rate can be charged without violating a credit limit. When a credit limit is exceeded, service can be denied or a message sent.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

Figure 23 illustrates some of the tables stored in an RDBMS for the purpose of storing real time account balance information. A relational database table (Account_T) 2301 is used to store general account information. Table 2301 includes an Account_Id column that identifies the account. A Name column identifies the name of the person that the account belongs to and a Next_Bill_Time column lists the time that the account is to be billed next. The use of the "next bill time" in connection with calculating billing at that end of a billing cycle and with a rating operation will be discussed in further detail below. Other fields describing the account are typically included in Account_T as well, and the Name field is shown only for the purpose of an example.

A relational database table (Account_Balance_T) 2303 contains information about various account balances. An Account_Id column identifies the account and may be utilized by a RDBMS to join the Account_T and Account_Balance_T tables. An Element_Id column identifies the balance element. A Current _Balance column contains the current balance for each account balance element and a Credit_Limit column stores the credit limit for each account balance element.

Figure 24 is a process flow diagram illustrating a regular billing process implemented on a computer system that accesses the account balance information in an RDBMS to generate billing for a billing cycle. The process runs sequentially on accounts.

At step 2401, an account being processed is locked. Locking the account creates a delay that prevents any of the account balances from being charged as a result of a billing event that occurs while the account is locked. In one embodiment locking the account is accomplished by setting a flag in the Account_T table. Next, at step 2403, the Next_Bill_Time column in the Account_T table is checked and determined if it is in the past at step 2405. Determining if the next bill time is in the past may include comparing the current time to the next bill time stored in the Account_T table.

If the next bill time is in the past then the system performs a close billing of the account at step 2407. Close billing of the account bills the account for any amounts due on the account during the billing cycle and it includes updating or advancing the next bill time in the Account_T table to the next billing cycle. Thus, close billing generates a bill for the account for the billing cycle.

The system unlocks the account at step 2409. If, at step 2405, it is determined that the next bill time is not in the past, then that is because the billing has already been performed and the next bill time was updated to a future time. In that case, control is transferred directly from step 2405 to step 2409.

The process of billing accounts typically operates sequentially on the accounts so at step 2411 it is determined if there is another account that needs to be processed. If there is, control returns to step 2401 and the next account is processed.

A rating operation determines a rate for the billing event and increments the various account balances impacted by the rate. When a transaction event that is billing occurs while accounts are being billed, it is possible that the account impacted by the billing event could be changed before it is closed out and billed. This would prevent a clean accounting close from being achieved. It is also possible that an account could have a billing event occur and have a rating operation occurring at the same time as a bill is being generated for the account. Such an occurrence might produce an unpredictable result. To prevent both of these problems, a special rating operation procedure is implemented. Whenever a billing event occurs, close billing is performed for the account impacted by the event if close billing has not already been performed. This so called event driven billing occurs out of sequence with respect to regular billing.

Figure 25 is a process flow diagram illustrating the process that occurs when a billing event occurs while the accounts are being billed. At step 2501, the system receives a transaction or billing event during the billing process. The account impacted by the billing event is locked at step 2503. Locking the account delays the regular billing process shown in Figure 24 from running on the account while the account is locked. Next, at step 2505, the next bill time is checked for the account to determine if it is in the past at step 2507. If the next bill time is in the past, then close billing is performed for the account and the next bill time is updated to the time (or date) of the next billing cycle at step 2509.

At step 2511, the balance impact(s) of the billing event are calculated and the balance elements of the account are updated. Thus, the billing event is posted to the account. If the next bill time is not in the past, then control is transferred directly from step 2507 to step 2511. At step 2513, the account is unlocked.

Thus, when a billing event occurs for an account occurs during a period when close billing is being run for the accounts, the next bill time is checked to determine whether billing has yet occurred for the account. If a bill has not yet been generated for the account, then billing is run immediately before the account balances are incremented. The interlocking feature between the processes shown in Figures 29 and 30 prevents regular billing from running on the account when event driven billing is running and vice versa.

The next bill time is analyzed to see if it is in the past in order to determine if an account has already been processed or billed during a close billing. Other mechanisms for determining whether an account has been billed during a close billing may be utilized. For example, a record

of the billing cycle where an account was last billed may also be utilized. Accordingly, the invention is not limited to the preferred embodiment described above.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

CLAIMS

1. In a computer system, a method of storing objects in a relational database, comprising the steps of:

defining a class of objects that are to be stored in the relational database, the objects of the class having at least one data member;

mapping each data member to at least one column in at least one relational database table; and

creating the at least one relational database table to store the objects of the class.

2. The method of claim 1, wherein the defining step comprises the step of generating a hierarchical tree that represents a definition of the data members of the class of objects.

3. The method of claim 1, wherein the hierarchical tree stores a hierarchy of classes including the class.

4. The method of claim 1, wherein the creating step comprises the steps of:

generating SQL statements for creating the at least one relational database table; and

sending the SQL statements to a relational database management system.

5. The method of claim 1, further comprising the step of instantiating an object of the class by instantiating a container object to store data members of the object.

6. The method of claim 5, wherein the container object is created utilizing a hierarchical tree that represents a definition of the data members of the class of objects.

7. The method of claim 1, further comprising the step of defining a subclass of objects that are to be stored in the relational database, the objects of the subclass inheriting the data members of the class of objects and having at least one additional data member.

8. The method of claim 7, further comprising the step of generating a hierarchical tree that comprises a definition of the at least one additional data member.

9. The method of claim 7, further comprising the steps of:

generating SQL statements for creating an additional relational database table to store the at least one additional data member; and

sending the SQL statements to a relational database management system.

10. The method of claim 1, wherein the at least one data member of the objects of the class comprise an array.

11. The method of claim 1, wherein the defining step comprises the step of receiving input of characteristics of data members.

12. The method of claim 1, wherein a characteristic indicates the data member is mandatory, optional, or read only.

13. The method of claim 1, wherein a characteristic indicates a default value for the data member.

14. In a computer system, a method of querying a relational database, comprising the steps of:

receiving a first query that is object-oriented and specifies information about objects of interest;

instantiating a query container object that comprises a query template based on the first query, an array for any arguments in the template query, and an array for any results in the template query;

utilizing the query container object, translating the first query into a second query in a relational database query language for accessing the specified information about the objects of interest that are stored by a relational database management system;

sending the second query to the relational database management system; and

receiving the specified information about the objects of interest from the relational database management system.

15. The method of claim 14, further comprising the step of utilizing a hierarchical tree to map data members of objects of interest to columns in the relational database.

16. The method of claim 14, further comprising the step of instantiating a results container object to store the specified information about the objects of interest.

17. In a computer system, a method of deferring allocation of storage for array elements of objects comprising the steps of:

receiving a request to instantiate an object of a class where the class has a definition that specifies a default value for each data member of an array element;

allocating storage space for the object without storage space for an array element if the instantiation request does not specify an initial value for any of the data members of the array element;

receiving a request to modify a data member of the array element;

determining if storage space for the array element has been allocated;

if storage space for the array element has not been allocated, allocating storage space for the array element and initializing each data member of the array element to the specified default value; and

modifying the data member of the array element as specified in the modification request.

18. The method of claim 17, wherein each array element is stored as a row in a relational database table.

19. A computer system for storing objects in a relational database, comprising:

an object-oriented application for receiving a definition of a class of objects that are to be stored in the relational database, the objects being stored in a temporary storage;

a memory for the temporary storage of the objects;

an object server for retrieving the objects from the memory and issuing statements in a relational database query language to store data of the objects; and

a relational database management system for receiving the statements and storing the data of the objects in persistent storage as relational database tables.

20. The system of claim 19, further comprising a hierarchical tree that represents the definition of the class of objects including data members.

21. The system of claim 20, wherein the hierarchical tree stores a mapping from each data member of the class of objects to at least one column in the relational database tables.

22. The system of claim 20, wherein the hierarchical tree stores a hierarchy of classes including the class.

23. A method of assigning payment resources to an account balance stored in an RDBMS table comprising:

instantiating a balance adjusting container object that includes an account element identification number, a balance element identification number of a balance element that will be adjusted and an adjusting amount;

formulating an RDBMS query to determine whether an entry in an account balance table exists that corresponds to the account element identification number and the balance element identification number;

if an entry in the account balance table is found, then adjusting the found entry in the account balance table by the adjusting amount;

if no entry in the account balance table is found, then creating an entry in the account balance table with a default balance and incrementing the created entry by the adjusting amount.

24. A method as recited in claim 23 wherein the default balance is zero.

25. A method as recited in claim 23 wherein the RDBMS query is an SQL query.

26. A method as recited in claim 23 further including sending the balance adjusting container to and object server and wherein the object server formulates the RDBMS query.

27. A method as recited in claim 23 wherein the object server formulates the RDBMS query using an hierarchical tree.

28. A method of defining a payment resource for inclusion in a real time transaction processing system comprising:

instantiating a balance element creating container object;

including data fields in the balance element creating container object including a balance element identification number field and providing a balance element identification number in the balance element identification number field.

adding a new row to a payment resources RDBMS table using the information from the balance element creating container object.

29. A method as recited in claim 28 wherein an object server determines the new row that is to be added in the payment resources RDBMS table.

30. A method as recited in claim 29 wherein the object server checks the balance element creating container object to make certain that the balance element creating container contains all of a set of required fields.

31. A method as recited in claim 28 wherein including data fields in the balance element creating container object further includes including a resource name field and a rounding amount field in the balance element creating container object.

32. A method of determining a rate for adjusting a real time balance including:

receiving a billing event, the billing event including a rate name, a quantity used, an attribute that describes the event, and an account to be billed;

searching a product table to determine all of the products in the account to be billed;

searching a rate table to determine all of the rates that correspond to the rate name and the products in the account to be billed;

eliminating rates that do not match the attribute that describes the event;

ordering the remaining rates by descending priority to obtain a rate stack; and

for each rate in descending order of priority, allocating the maximum portion of the quantity used that can be applied using the rate without violating a credit limit.

33. A method as recited in claim 32 further including applying a default rate to a portion of the quantity used that cannot be applied using one of the remaining rates without violating a credit limit.

34. In a computer system, a method of performing real time billing of accounts, comprising the steps of:

during close billing of a billing cycle, receiving a transaction event for an account;

locking the account;

determining if the account has already been billed in the billing cycle;

if the account has not already been billed in the billing cycle, close billing the account without billing the transaction event;

posting the transaction event to the account; and

unlocking the account.

35. The method of claim 34, wherein the determining step includes the step of determining if the next bill time is in the past.

36. The method of claim 35, wherein the billing step includes the step of advancing the next bill time of the account to a next billing cycle.

37. The method of claim 35, wherein the next bill time is compared to a current time.

38. The method of claim 34, wherein the transaction event is a billing event.

39. In a computer system, a method of performing real time billing of accounts, comprising the steps of:

during close billing of a billing cycle, receiving a billing event for an account;

locking the account;

determining if the account has already been billed in the billing cycle by checking if a next bill time of the account is in the past;

if the account has not already been billed in the billing cycle, close billing the account without billing the billing event;

posting the billing event to the account; and

unlocking the account.

40. The method of claim 39, wherein the billing step includes the step of advancing the next bill time of the account to a next billing cycle.

41. The method of claim 39, wherein the next bill time is compared to a current time.

42. A computer implemented real time billing system, comprising:

a processor;

a memory coupled to the processor that stores accounts;

a billing process operating on the processor that bills accounts at the end of a billing cycle and upon receiving a transaction event for an account, the billing process locks the account, close bills the account without billing the transaction event if the account has not already been billed in the billing cycle, posts the transaction event to the account, and unlocks the account.

43. The system of claim 42, wherein it is determined if the account has not already been billed in the billing cycle if a next bill time is in the past.

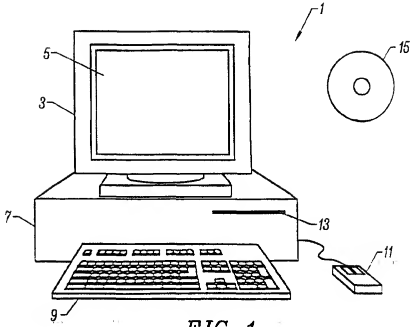

1
5
3
15
7
13
11
9

FIG. 1

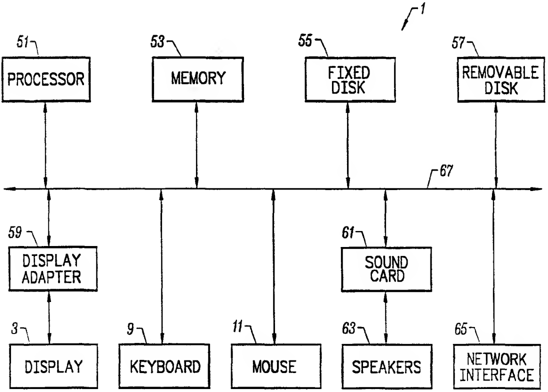

1
51
PROCESSOR
53
MEMORY
55
FIXED DISK
57
REMOVABLE DISK
67
59
DISPLAY ADAPTER
61
SOUND CARD
3
DISPLAY
9
KEYBOARD
11
MOUSE
63
SPEAKERS
65
NETWORK INTERFACE

FIG. 2

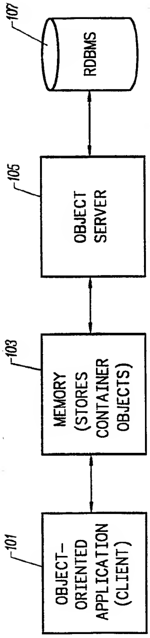

107
RDBMS
105
OBJECT SERVER
103
MEMORY (STORES CONTAINER OBJECTS)
101
OBJECT-ORIENTED APPLICATION (CLIENT)

FIG. 3

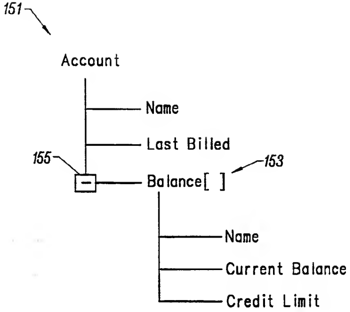

151
Account
Name
Last Billed
155
Balance[ ]
153
Name
Current Balance
Credit Limit

FIG. 4

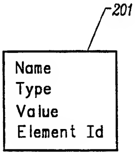
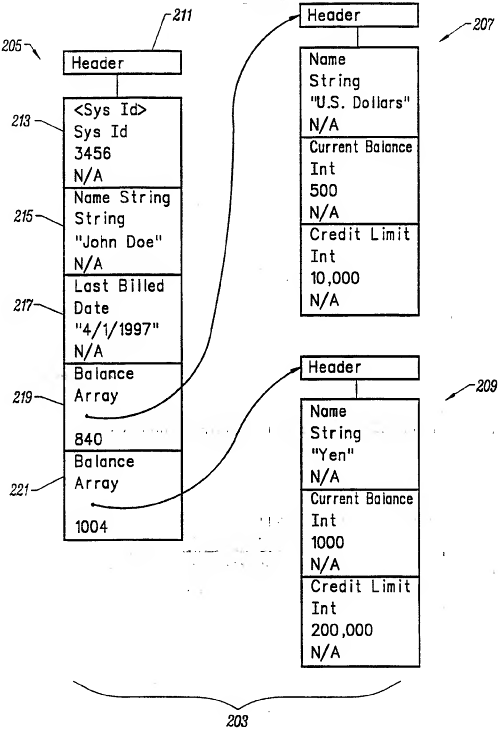

201
Name
Type
Value
Element Id

FIG. 5A



205
211
Header
<Sys Id>
Sys Id
3456
N/A
213
Name String
String
"John Doe"
N/A
215
Last Billed
Date
"4/1/1997"
N/A
217
Balance
Array
840
219
Balance
Array
1004
221
Header
207
Name
String
"U.S. Dollars"
N/A
Current Balance
Int
500
N/A
Credit Limit
Int
10,000
N/A
Header
209
Name
String
"Yen"
N/A
Current Balance
Int
1000
N/A
Credit Limit
Int
200,000
N/A
203

FIG. 5B

Account_T 251

Id	Name	Last_Billed
3456	John Doe	"4/1/1997"
⋮		

Account_Balance_T 253

Id	Elemt_Id	Name	Current_Balance	Credit_Limit
3456	840	U.S. Dollars	500	10,000
3456	1004	Yen	1000	200,000
⋮				

FIG. 6

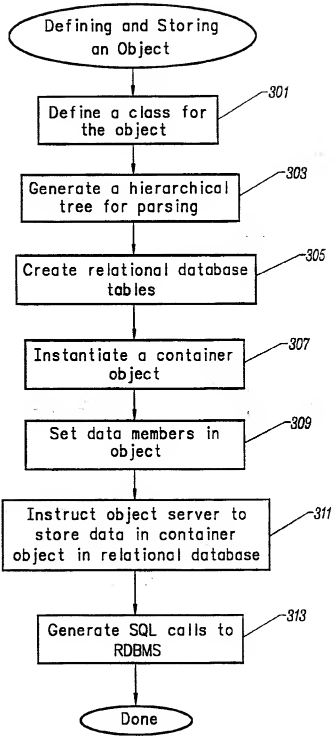

Defining and Storing an Object
Define a class for the object
301
Generate a hierarchical tree for parsing
303
Create relational database tables
305
Instantiate a container object
307
Set data members in object
309
Instruct object server to store data in container object in relational database
311
Generate SQL calls to RDBMS
313
Done

FIG. 7

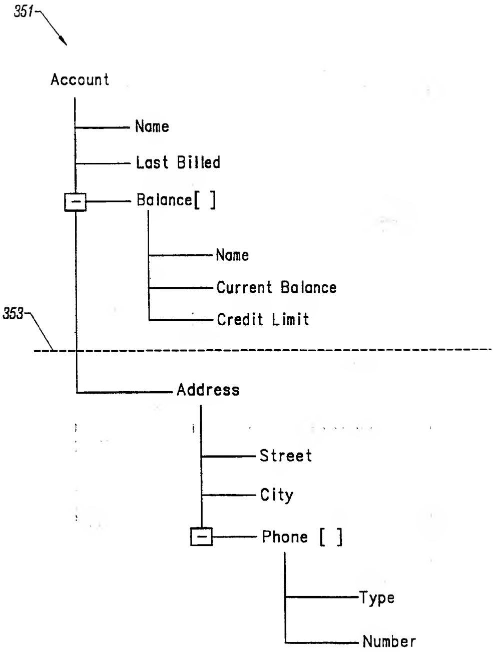

351
Account
Name
Last Billed
Balance[ ]
Name
Current Balance
Credit Limit
353
Address
Street
City
Phone [ ]
Type
Number

FIG. 8

401

Account_Address_T

Id	Street	City
3456	Elm St.	San Jose
. . .		

403

Account_Address_Phone_T

Id	Element_Id	Type	Number
3456	56	Home	555-1111
3456	58	Work	555-1112
. . .			

FIG. 9

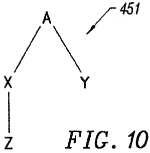

451
A
X
Y
Z

FIG. 10

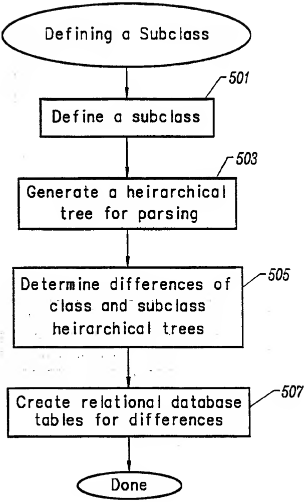

Defining a Subclass
Define a subclass
501
Generate a heirarchical tree for parsing
503
Determine differences of class and subclass heirarchical trees
505
Create relational database tables for differences
507
Done

FIG. 11

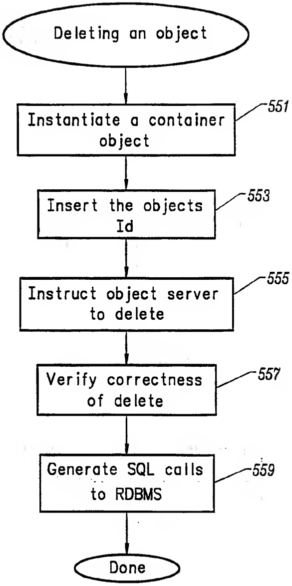

Deleting an object
Instantiate a container object
551
Insert the objects Id
553
Instruct object server to delete
555
Verify correctness of delete
557
Generate SQL calls to RDBMS
559
Done

FIG. 12

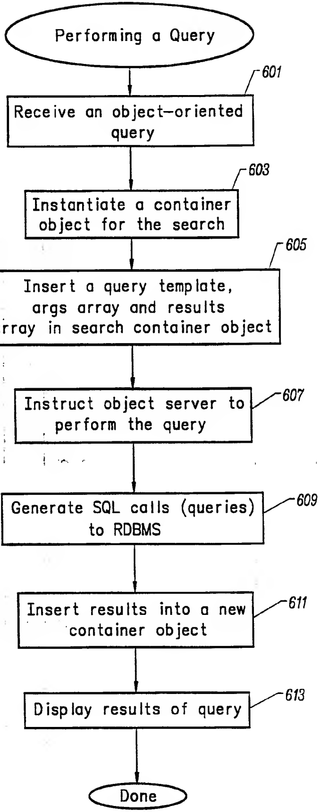

Performing a Query
601
Receive an object-oriented query
603
Instantiate a container object for the search
605
Insert a query template, args array and results array in search container object
607
Instruct object server to perform the query
609
Generate SQL calls (queries) to RDBMS
611
Insert results into a new container object
613
Display results of query
Done

FIG. 13

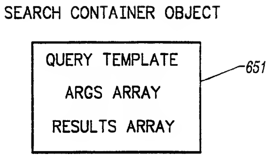
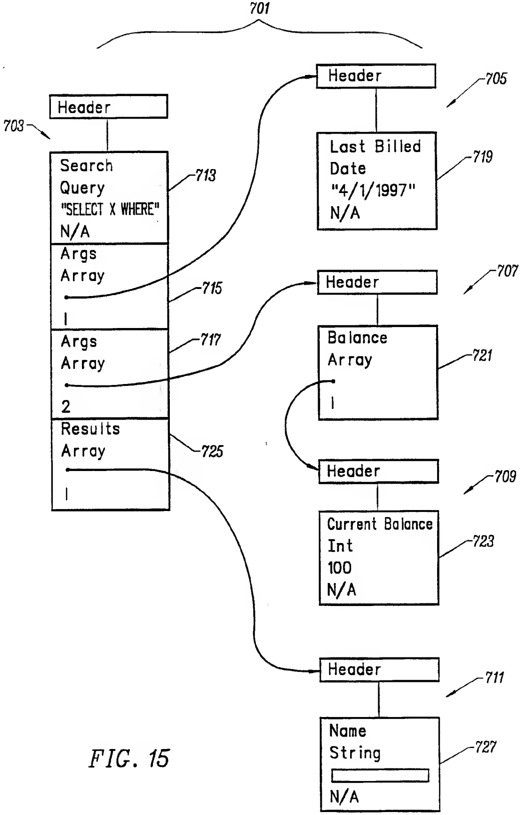

SEARCH CONTAINER OBJECT
QUERY TEMPLATE
ARGS ARRAY
RESULTS ARRAY
651

FIG. 14



701
Header
703
Search
Query
"SELECT X WHERE"
N/A
713
Args
Array
1
715
Args
Array
2
717
Results
Array
1
725
Header
705
Last Billed
Date
"4/1/1997"
N/A
719
Header
707
Balance
Array
1
721
Header
709
Current Balance
Int
100
N/A
723
Header
711
Name
String
N/A
727

FIG. 15

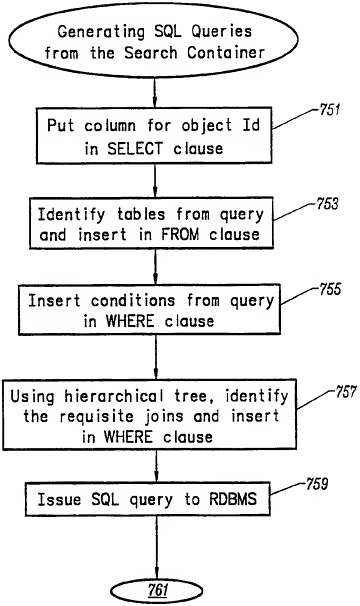

Generating SQL Queries from the Search Container
Put column for object Id in SELECT clause
751
Identify tables from query and insert in FROM clause
753
Insert conditions from query in WHERE clause
755
Using hierarchical tree, identify the requisite joins and insert in WHERE clause
757
Issue SQL query to RDBMS
759
761

FIG. 16A

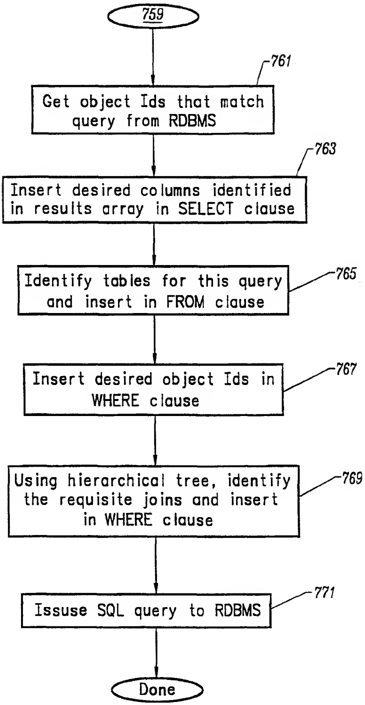

759
761
Get object Ids that match query from RDBMS
763
Insert desired columns identified in results array in SELECT clause
765
Identify tables for this query and insert in FROM clause
767
Insert desired object Ids in WHERE clause
769
Using hierarchical tree, identify the requisite joins and insert in WHERE clause
771
Issuse SQL query to RDBMS
Done

FIG. 16B

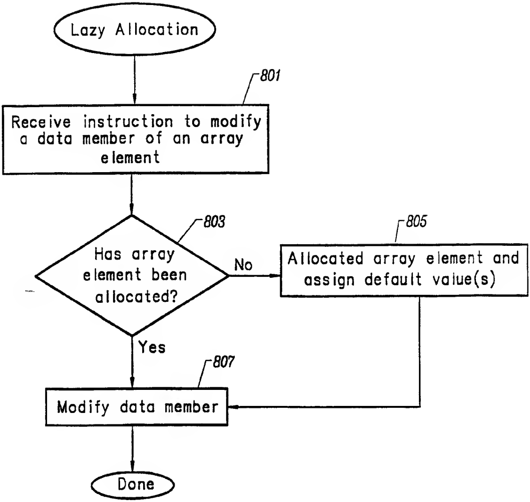

Lazy Allocation
801
Receive instruction to modify a data member of an array element
803
Has array element been allocated?
No
805
Allocated array element and assign default value(s)
Yes
807
Modify data member
Done

FIG. 17

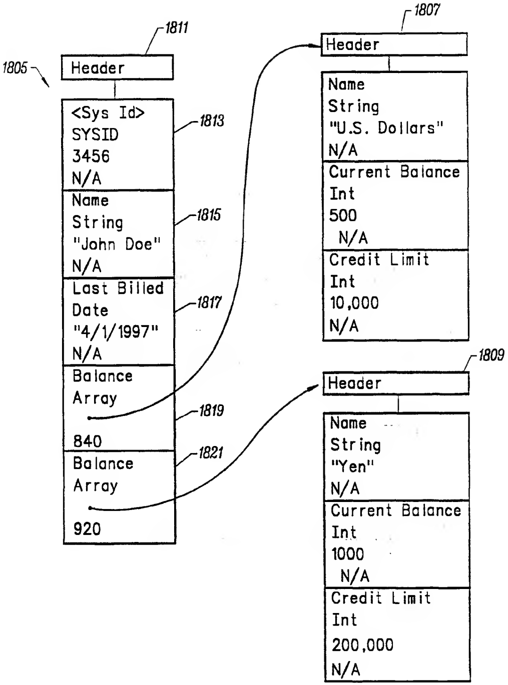

1811
1805
Header
<Sys Id>
SYSID
3456
N/A
1813
Name
String
"John Doe"
N/A
1815
Last Billed
Date
"4/1/1997"
N/A
1817
Balance
Array
840
1819
Balance
Array
920
1821
1807
Header
Name
String
"U.S. Dollars"
N/A
Current Balance
Int
500
N/A
Credit Limit
Int
10,000
N/A
1809
Header
Name
String
"Yen"
N/A
Current Balance
Int
1000
N/A
Credit Limit
Int
200,000
N/A

FIG. 18A

Account_T 1851

Sys_Id	Id	Name	Last_Billed
<Sys Id>	3456	John Doe	"4/1/1997"
⋮			

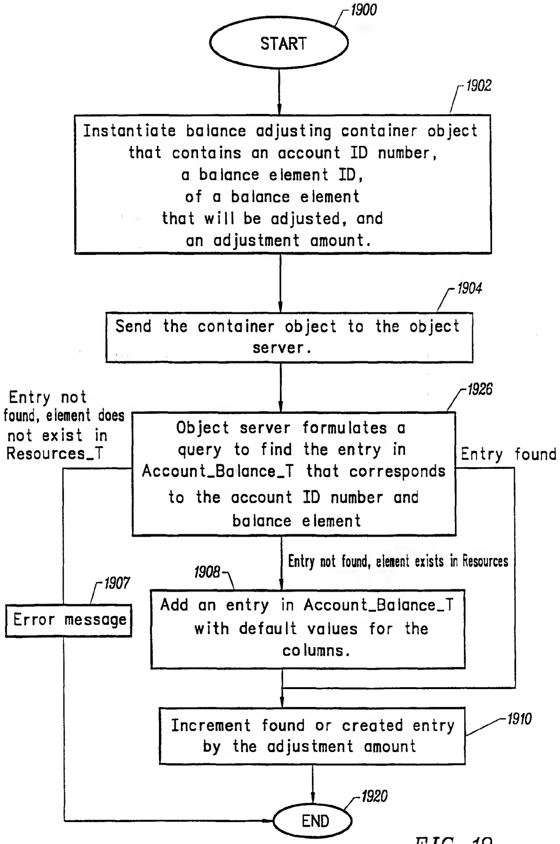
Account_Balance_T 1853

Id	Element_Id	Current_Balance	Credit_Limit
3456	840	500	10,000
3456	1004	1000	200,000
⋮			

Resources_T 1854

Balance Element Id	Resource Name	GUI Name	Rounding
840	U.S. dollars		.01
920	Yen		.01
1,000,001	Frequent Flyer Miles		1.

FIG. 18B



1900
START
1902
Instantiate balance adjusting container object that contains an account ID number, a balance element ID, of a balance element that will be adjusted, and an adjustment amount.
1904
Send the container object to the object server.
1926
Object server formulates a query to find the entry in Account_Balance_T that corresponds to the account ID number and balance element
Entry not found, element does not exist in Resources_T
Entry found
Entry not found, element exists in Resources
1907
Error message
1908
Add an entry in Account_Balance_T with default values for the columns.
1910
Increment found or created entry by the adjustment amount
1920
END

FIG. 19

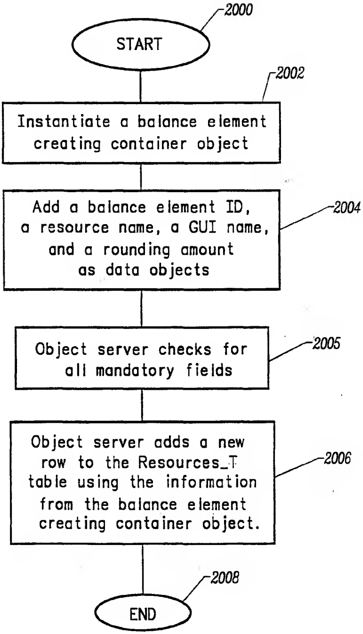

2000
START
2002
Instantiate a balance element creating container object
Add a balance element ID, a resource name, a GUI name, and a rounding amount as data objects
2004
Object server checks for all mandatory fields
2005
Object server adds a new row to the Resources_T table using the information from the balance element creating container object.
2006
2008
END

FIG. 20

Product_t

PRODUCT ID	NAME	ATTRIBUTES
6492	PRODUCT A PRODUCT B PRODUCT C	

2102

RATE_t

RATE ID	PRODUCT_ID	NAME	PRIORITY	DEFAULT FLAG	ATTRIBUTES
1234	6492 6492 6492	TYPE1 TYPE2		1	

2104

ACCT_t

ACCT_ID	<ACCOUNT ATTRIBUTES>

2106

ACCT_BAL_t

ACCOUNT ID	ELEMENT ID	CURR BAL	CREDIT LIMIT

2108

ACCT_PROD_t

ACCOUNT ID	ELEMENT ID	PRODUCT ID

2110

RATE_BAL_IMPACT_t

RATE ID	ELEMENT ID	BAL_ELEMENT	FIXED OPERAND	FREE QUANTITY	TYPE
1234 1234 1234	1 2 3	840 1204 840			

2112

FIG. 21

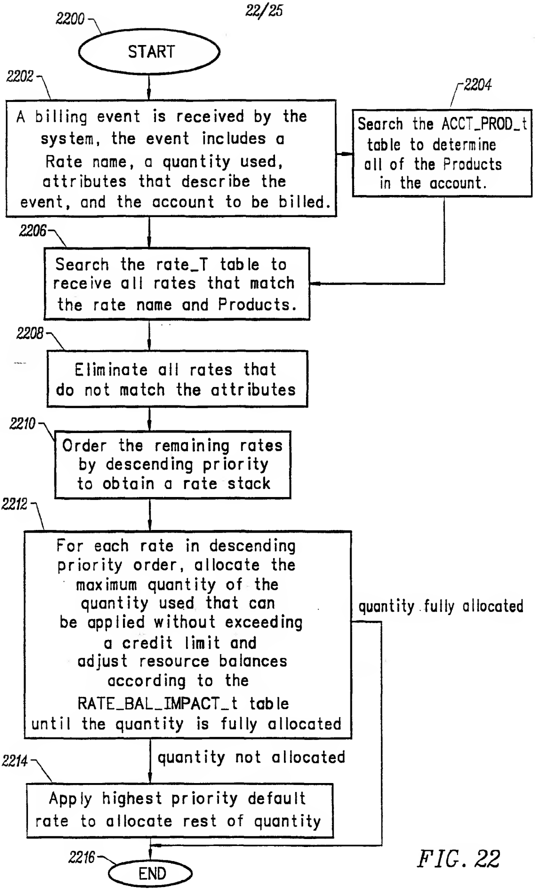

2200
START
2202
A billing event is received by the system, the event includes a Rate name, a quantity used, attributes that describe the event, and the account to be billed.
2204
Search the ACCT_PROD_t table to determine all of the Products in the account.
2206
Search the rate_T table to receive all rates that match the rate name and Products.
2208
Eliminate all rates that do not match the attributes
2210
Order the remaining rates by descending priority to obtain a rate stack
2212
For each rate in descending priority order, allocate the maximum quantity of the quantity used that can be applied without exceeding a credit limit and adjust resource balances according to the RATE_BAL_IMPACT_t table until the quantity is fully allocated
quantity fully allocated
quantity not allocated
2214
Apply highest priority default rate to allocate rest of quantity
2216
END

FIG. 22

2301

ACCT_T

Account_Id	Name	Next_Bill_Time

2303

Account_Balance_T

Account_Id	Element_Id	Current_ Balance	Credit_ Limit

FIG. 23

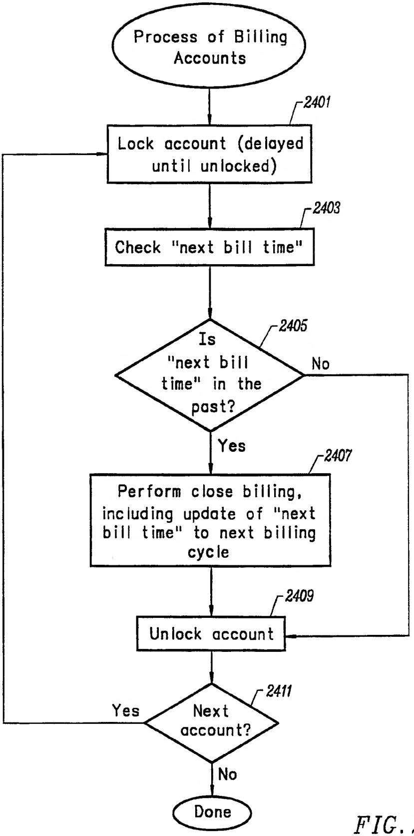

Process of Billing Accounts
2401
Lock account (delayed until unlocked)
2403
Check "next bill time"
2405
Is "next bill time" in the past?
No
Yes
2407
Perform close billing, including update of "next bill time" to next billing cycle
2409
Unlock account
2411
Next account?
Yes
No
Done

FIG. 24

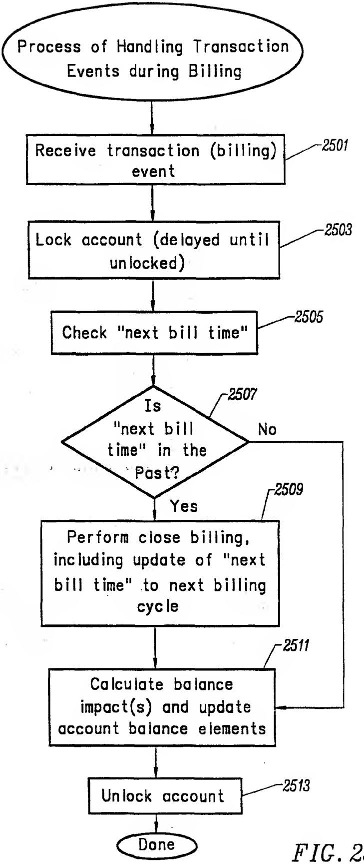

Process of Handling Transaction Events during Billing
Receive transaction (billing) event
2501
Lock account (delayed until unlocked)
2503
Check "next bill time"
2505
Is "next bill time" in the Past?
2507
No
Yes
Perform close billing, including update of "next bill time" to next billing cycle
2509
Calculate balance impact(s) and update account balance elements
2511
Unlock account
2513
Done

FIG. 25

INTERNATIONAL SEARCH REPORT

Inte ional Application No
PCT/US 98/10116

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30 G06F17/60

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 96 34350 A (ASPECT DEV INC) 31 October 1996 see page 4, line 12 - page 9, line 20 see page 23, line 20 - page 25, line 18 see page 31, line 18 - page 31, line 22 see page 47, line 7 - page 47, line 8 see figures 1,2	1-22
Y		23-43

X	WO 97 03406 A (WALL DATA INC ;KAWAI KENJI (US)) 30 January 1997 see abstract see page 23, line 28 - page 24, line 7	1,4, 19-22

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publicationdate of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of theinternational search	Date of mailing of the international search report
15 October 1998	21/10/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Abbing, R

2

Inte. onal Application No
PCT/US 98/10116

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	CHENHO KUNG: "OBJECT SUBCLASS HIERARCHY IN SQL: A SIMPLE APPROACH" COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, vol. 33, no. 7, 1 July 1990, pages 117-125, XP000143088 see the whole document ---	1,7-9
A	US 5 530 853 A (SCHELL DAVID J ET AL) 25 June 1996 see column 1, line 13 - column 2, line 10 see claims ---	1,5,6, 14-16
Y	WO 95 27255 A (REGAN TIMOTHY ;WILSON JEREMY (GB); FREESTONE DAVID (GB); BRITISH T) 12 October 1995 see the whole document	23-43
A	---	1-22
A	WO 95 04960 A (PERSISTENCE SOFTWARE INC) 16 February 1995 see abstract see page 5, line 33 - page 7, line 2 -----	19-22

2

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte ional Application No
PCT/US 98/10116

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9634350 A	31-10-1996	EP 0823092 A	11-02-1998
WO 9703406 A	30-01-1997	US 5717924 A	10-02-1998
		AU 6251796 A	10-02-1997
		CA 2222583 A	30-01-1997
		EP 0838060 A	29-04-1998
		NO 980068 A	06-01-1998
US 5530853 A	25-06-1996	JP 6202918 A	22-07-1994
		JP 8020982 B	04-03-1996
WO 9527255 A	12-10-1995	SG 43130 A	17-10-1997
		AU 2080095 A	23-10-1995
		CA 2186626 A	12-10-1995
		EP 0753179 A	15-01-1997
WO 9504960 A	16-02-1995	US 5615362 A	25-03-1997
		US 5706506 A	06-01-1998